Evolutionary Control of Autonomous Underwater Vehicles

A thesis submitted in fulfilment of the requirements for

the degree of Doctor of Philosophy

Royce R. Smart

B.Eng, B.App.Sc

School of Aerospace, Mechanical and Manufacturing Engineering

Science, Engineering and Technology Portfolio

RMIT University

August 2008

Declaration

I certify that except where due acknowledgement has been made, the work is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; the content of the thesis is the result of work which has been carried out since the official commencement date of the approved research program; and, any editorial work, paid or unpaid, carried out by a third party is acknowledged.

Signed

Royce R. Smart

August 2008

Acknowledgements

Firstly, I would like to thank my supervisors, John Wharington and Pavel Trivailo, for their advice and patience. I am especially grateful for the support and friendship of John Wharington during these difficult years.

I would like to extend a special thanks to all those who made me feel so welcome during my time at the Maritime Platforms Division of the Defence Science and Technology Organisation. My involvement with the unmanned underwater vehicles program has been an invaluable experience. In particular, I would like to thank Francis Valentinis for his support and friendship. I would also like to thank Roger Neil and Kevin Gaylor for their support.

Andrew Walker provided the software used to generate the optimal Dubins Curves in Chapter 6 and kindly agreed to read the draft of my thesis. I am also grateful for his friendsip and our many interesting conversations about robotics.

This research was funded in part by an Australian Postgraduate Award and a top-up scholarship from the Defence Science and Technology Organisation.

Finally, I would like to thank Dorothy without whose love and support none of this would have been possible.

Royce Smart

Contents

List of Figures

List of Tables

List of acronyms

AGE	Analog Genetic Encoding
ANN	Artificial Neural Network
AUV	Autonomous Underwater Vehicle
BBR	Behaviour-Based Robotics
CMA-ES	Covariance Matrix Adaptation Evolution Strategy
CoSyNE	Cooperative Synapse Neuroevolution
EA	Evolutionary Algorithm
EANT	Evolutionary Acquisition of Neural Topologies
EC	Evolutionary Computation
EP	Evolutionary Programming
ER	Evolutionary Robotics
ES	Evolution Strategies
ESP	Enforced SubPopulations
GA	Genetic Algorithm
GP	Genetic Programming
GRN	Gene Regulatory Network

IMU	Inertial Measurement Unit
LQR	Linear-Quadratic Regulator
MSC	Mutative Strategy Parameter Control
NE	Neuroevolution
NEAT	NeuroEvolution of Augmenting Topologies
NFL	No Free Lunch
NN	Neural Network
ODE	Open Dynamics Engine
RL	Reinforcement Learning
RNN	Recurrent Neural Network
SANE	Symbiotic, Adaptive Neuro-Evolution
SDRE	State Dependent Riccati Equation
TD	Temporal Difference
UAV	Unmanned Aerial Vehicle

Abstract

The goal of Evolutionary Robotics (ER) is the development of automatic processes for the synthesis of robot control systems using evolutionary computation. The idea that it may be possible to synthesise robotic control systems using an automatic design process is appealing. However, ER is considerably more challenging and less automatic than its advocates would suggest.

ER applies methods from the field of neuroevolution to evolve robot control systems. Neuroevolution is a machine learning algorithm that applies evolutionary computation to the design of Artificial Neural Networks (ANN). The aim of this thesis is to assay the practical characteristics of neuroevolution by performing bulk experiments on a set of Reinforcement Learning (RL) problems. This thesis was conducted with the view of applying neuroevolution to the design of neurocontrollers for small low-cost Autonomous Underwater Vehicles (AUV).

A general approach to neuroevolution for RL problems is presented. The Covariance Matrix Adaptation Evolution Strategy (CMA-ES) is selected to evolve ANN connection weights on the basis that it has shown competitive performance on continuous optimisation problems, is self-adaptive and can exploit dependencies between connection weights. Practical implementation issues are identified and discussed.

A series of experiments are conducted on RL problems. These problems are representative of problems from the AUV domain, but manageable in terms of problem complexity and computational resources required. Results from these experiments are analysed to draw out practical characteristics of neuroevolution.

Bulk experiments are conducted using the inverted pendulum problem. This popular control benchmark is inherently unstable, underactuated and non-linear: characteristics common to underwater vehicles. Two practical characteristics of neuroevolution are demonstrated: the

importance of using randomly generated evaluation sets and the effect of evaluation noise on search performance. As part of these experiments, deficiencies in the benchmark are identified and modifications suggested.

The problem of an underwater vehicle travelling to a goal in an obstacle free environment is studied. The vehicle is modelled as a Dubins car, which is a simplified model of the high-level kinematics of a torpedo class underwater vehicle. Two practical characteristics of neuroevolution are demonstrated: the importance of domain knowledge when formulating ANN inputs and how the fitness function defines the set of evolvable control policies. Paths generated by the evolved neurocontrollers are compared with known optimal solutions.

A framework is presented to guide the practical application of neuroevolution to RL problems that covers a range of issues identified during the experiments conducted in this thesis. An assessment of neuroevolution concludes that it is far from automatic yet still has potential as a technique for solving reinforcement problems, although further research is required to better understand the process of evolutionary learning.

The major contribution made by this thesis is a rigorous empirical study of the practical characteristics of neuroevolution as applied to RL problems. A critical, yet constructive, viewpoint is taken of neuroevolution. This viewpoint differs from much of the reseach undertaken in this field, which is often unjustifiably optimistic and tends to gloss over difficult practical issues.

CHAPTER 1

Introduction

During the past decade the cost of building robotic platforms has decreased significantly due to the availability of low-cost sensors and hardware. This has seen an increase in the commercial availability of mobile robots for "dull, dirty and dangerous" tasks in the home and on the battlefield. However, the design of control systems that enable such robots to autonomously undertake useful tasks in unstructured environments remains a challenging problem.

Two related factors make the the design of control systems for mobile robots challenging. Firstly, mobile robots operate in unstructured environments that are inherently unpredictable and dynamical. A mobile robot must, therefore, be capable of dealing with the uncertainty that is present in the many possible interactions between it and the world. Secondly, a mobile robot must possess some degree of *intelligence* if it is to undertake tasks without direct human involvement in such environments. No widely agreed upon scientific definition of intelligence exists, but it can be described generally as an open collection of related cognitive abilities. Of these abilities, those most relevant to mobile robotics are associated with autonomy and include abilities such as perception, adaptation, and planning.

The idea that it may be possible to synthesise robot control systems via an automatic design process is very appealing. An automatic design process can reduce the degree of knowledge that a human designer requires about the complex interactions between a robot and its environment. Also, an automatic design process is able to find novel solutions in regions of the design space excluded by design constraints imposed by a human designer.

The development of automatic processes for the synthesis of robot control systems using evolutionary computation is the primary goal of Evolutionary Robotics (ER). ER applies methods from the field of Evolutionary Computation (EC) to evolve a population of robot control systems. EC encompasses a range of stochastic optimisation techniques inspired by the

process of biological evolution.

The general method followed in ER can be described as follows. An initial population of robot control systems are randomly generated. Each control system is evaluated on an embodied robot in an environment where the robot is free to act as directed by its control system. The high-level task performance of each robot is used to select the best control systems, which are advanced to the next generation with variations introduced by stochastic operators. This generational process of evaluation, selection and variation is repeated until a control system emerges that satisfies the high-level task performance requirements.

Despite the aspirations of ER, evolutionary methods are considerably less automatic and more challenging than their advocates would sometimes suggest. Tellingly, research in the field of ER has been mostly restricted to proof of concept experiments using simple robots in simple environments. No research has demonstrated evolved robot control systems that can significantly outperform those designed by a human designer.

1.1 Aim

The work in this thesis was conducted with the view of applying evolutionary methods to the design of neurocontrollers for small low-cost Autonomous Underwater Vehicles (AUV). Such vehicles can be characterised by limited and noisy sensing capabilities, limited computational resources and limited energy sources. Although a single small low-cost AUV may have limited utility for some tasks, a group of such vehicles could possibly outperform a single larger, more expensive vehicle. The ER approach has the potential to develop robust control systems for these vehicles and to find novel solutions that make the best use of the limited sensing, computational and energy resources available. Also, the trend towards the commoditisation of robots and the emergence of nanorobotics is a motivating factor for robotics research in this field.

Evolved control systems in ER are typically implemented as Artificial Neural Networks (ANN), which are computational models of the biological neural networks that make up the brain. ANNs are capable of expressing general non-linear functions, can tolerate the presence of noise in input data and permit the application of evolution at several different levels. These properties make ANNs well suited for use in ER.

The application of EC to the design of ANNs is the domain of a field of study known as Neuroevolution (NE). Even though NE is inspired by biological evolution, it is applied to problems as a machine learning algorithm. Machine learning is a broad field of study that is concerned with "programing computers to optimise a performance criterion using example data or past experience" (Alpaydin, 2004). It is for this reason that the process of NE can be described as *evolutionary learning*.

NE shares many common characteristics with another machine learning algorithm known as Reinforcement Learning (RL) and has been primarily applied to the RL class of problems. A RL problem involves an agent learning through interaction with the environment. The agent, being a system that can observe and act upon its environment, must find a policy consisting of a sequence of actions that maximises some long-term reward. RL is relevant to robotics because it involves finding reasonable solutions to problems with minimal *a priori* knowledge.

The aim of this thesis is to assay the practical characteristics of neuroevolution by performing bulk experiments on a set of reinforcement learning problems.

The motivation for this research is that practical issues are a common cause of failure when applying NE methods in practice, but have received relatively little attention in the published literature. Thus, a rigorous study of the difficulties would be of value to practitioners.

1.2 Overview

The thesis is broadly organised as follows. The early chapters present relevant background material. They are followed by a chapter outlining the approach to NE presented in this thesis. This approach is then applied to individual control problems in the core chapters from which general principles of evolutionary learning are derived and discussed in the concluding chapters. A description of each chapter in the thesis is given below.

Chapter 2 presents background material relevant to this thesis. Firstly, the chapter describes EC and ANNs in sufficient technical detail to understand the work presented in the following chapters. Secondly, the chapter reviews NE and ER with emphasis on relevant methods, applications and challenges.

Lessons learnt by the author from early work conducted in the fields of NE and ER are briefly discussed in Chapter 3. This discussion provides additional context for the approach

followed in subsequent chapters and may prove useful to other researchers undertaking work in these areas.

Chapter 4 presents the general approach to NE that is followed in this thesis. The rationale behind the selected approach is explained and pertinent implementation issues are discussed. In particular, the cause and effects of evaluation noise in the fitness function are addressed and methods to reduce its influence are presented.

Results from bulk experiments using the inverted pendulum system are analysed in Chapter 5 to gain insight into the NE approach followed in this thesis. The inverted pendulum system is used in place of an AUV system for theses experiments so as to minimise the computational resources required and permit greater insight into the results. As part of these experiments, deficiencies were identified in the inverted pendulum problem benchmark and modifications to the benchmark are suggested.

In Chapter 6, the NE approach is applied to the problem of a vehicle travelling to a goal in an obstacle free environment. Issues related to the formulation of ANN inputs and fitness function design are discussed that serve to characterise practical aspects of neuroevolution. The paths generated by the best evolved ANNs are compared with known optimal solutions.

Chapter 7 presents a framework to guide the practical application of NE to RL problems. This framework is based on an analysis of the results from the experiments conducted in this thesis and covers seven critical issues.

The concluding chapter presents a summary of the major findings of the thesis. An assessment of NE concludes that it is far from automatic yet still has potential as a technique for solving RL problems, although further research is required to better understand the process of evolutionary learning.

Appendix A presents the derivation of the equations of motion for the dual inverted pendulum system studied in Chapter 5.

1.3 Outcomes

The major contribution made by this thesis is a rigorous empirical study of the practical characteristics of NE as applied to RL problems. A critical, yet constructive, viewpoint is taken of NE. This viewpoint differs from much of the reseach undertaken in this field, which is often

unjustifiably optimistic and tends to gloss over the difficult practical issues.

A framework is presented that compiles a range of issues relevant to the practical application of NE. This framework is based on an analysis of the results from bulk experiments conducted on a set of RL problems. These problems are representative problems from the AUV domain, yet by rigorously testing these problems practical characteristics of NE were identified that would have been otherwise obscured by more complex problems.

The framework is intended to serve two purposes. Firstly, it provides a guide for practitioners who are considering applying NE to RL problems. Secondly, it identifies issues that represent barriers to the widespread adoption of NE as a tool for solving RL problems and, thereby, suggests areas worthy of future research. However, many of these issues are unlikely to be solved simply by "tinkering" with algorithms. The empirical support presented in this thesis of the framework serves both to illustrate the issues and to signify their importance. Furthermore, this thesis explores to what degree common benchmark problems are suited as test problems for new algorithms.

Bulk experiments were conducted on the inverted pendulum problem. Even though this problem has been widely studied and used to benchmark nearly all NE methods, it is not evident that previous authors have critically analysed their results with the same degree of rigour as is shown in this thesis. In this work, deficiencies were identified in the inverted pendulum problem benchmark itself. Modifications to the benchmark are suggested and adopted.

The importance of using randomly generated evaluation sets to create sufficient selection pressure for the evolution of general solutions is demonstrated. The evaluation noise created by random evaluation sets is discussed in depth and a series of experiments using the inverted pendulum problem demonstrate its negative effects. General principles for dealing with evaluation noise are proposed and tested, and these show some amelioration of the problem.

Experiments were conduced on the Dubins car problem, a popular representation of mobile robot kinematics used in navigation tasks. From these experiments, practical characteristics of NE are demonstrated, which include the formulation of Neural Network (NN) inputs and fitness function design. The capability of NE methods to find optimal policies was explored by comparing the paths generated by the evolved neurocontrollers with the known optimal paths. It is shown that the evolved neurocontrollers approach optimal behaviour, but that they fail to learn optimal paths for the entire state space.

CHAPTER 2

Background

This chapter reviews background material relevant to the work presented in this thesis. Firstly, the general concept of evolutionary computation is introduced and evolution strategies are described in sufficient detail to provide a technical background for the work to follow. Neural networks are then briefly introduced before methods for the application of evolutionary algorithms to the design of neural networks are reviewed. This is followed by a comparison between learning and evolution as applied to neural networks. Finally, the chapter concludes with a review of evolutionary robotics and the challenges associated with the application of this approach.

2.1 Evolutionary computation

The field of Evolutionary Computation (EC) encompasses a range of stochastic optimisation techniques inspired by the process of biological evolution. Evolutionary Algorithms (EA) imitate natural selection, which is the process by which favourable traits become more common in successive generations of a population of reproducing organisms and unfavourable traits become less common. Many variants of EAs exist, but all follow the general method shown in Figure 2.1. For a comprehensive review of EAs see (Eiben and Smith, 2003).

EAs maintain a multiset or *population* of *individuals*. An individual contains a *genotype* that encodes a point in the search space. The elements of the genotype are known as variables or *genes*. A *phenotype* is the decoded version of the genotype in the problem domain. A direct encoding may exist between the genotype and phenotype such that the distinction between the two is unimportant.

The performance of an individual in the problem domain is represented by its *fitness* and is

measured using a *fitness function*. The fitness of an individual is used to rank it against other individuals in the population. Although the term fitness implies a maximisation problem, it can be equally applied to minimisation problems.

At initialisation of the EA the individuals in the population are randomly generated. Each individual is evaluated using the fitness function and assigned a fitness. Some of the fittest individuals are selected as *parents* and used to create new individuals or *children*. Children are created by two *variation* operators:

Recombination Recombination merges the genotypes of two or more parents to form one or more children.

Mutation Mutation creates a single child by stochastically modifying the genotype of a single parent.

The method used for parent selection and the role of the variation operators varies between different EAs. The fitness of each child is evaluated using the fitness function. Based on fitness, individuals or *survivors* are selected from the children and possibly parents for inclusion in the population of the next iteration or *generation*. The methods used to select parents and survivors are known as the *selection* operators. This generational process of selection and variation is repeated until the termination conditions are satisfied.

The EA family consists of four historical members that have emerged more or less independently (Eiben and Smith, 2003). They are distinguished from each other by how individuals are represented as well as the role and implementation of the variation and selection operators. However, the boundaries between the members are not always distinct. The four historical members of the EA family can be briefly described as follows:

Genetic Algorithms Genetic Algorithms (GA; Holland, 1992; Goldberg, 1989) are the most widely known of the four members of the EA family. They have been applied to both continuous optimisation and combinatorial optimisation problems. Individuals are represented by bit-strings that encode the values of the parameters being optimised.

Evolution Strategies Evolution Strategies (ES; Rechenberg, 1973; Schwefel, 1975) were designed for continuous optimisation problems. Individuals are represented by real-valued

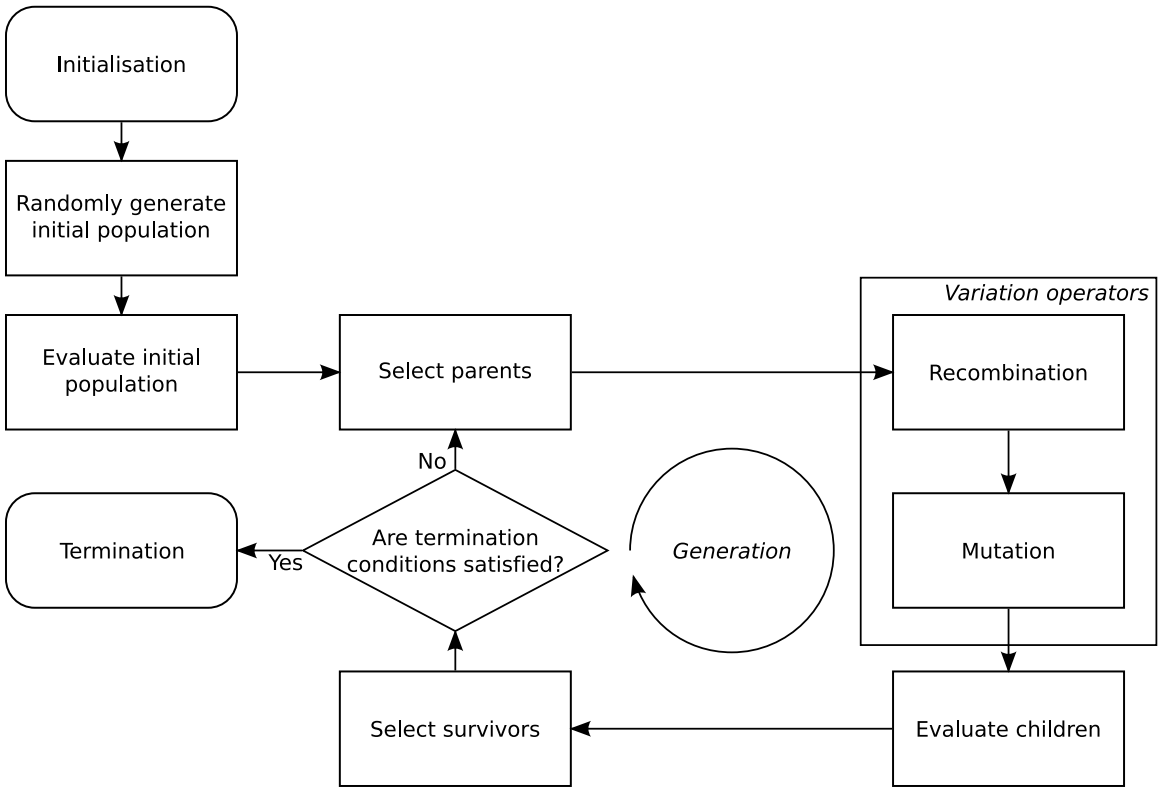


Figure 2.1: Flowchart of the general evolutionary algorithm method.

vectors and mutation is the primary variation operator. ES are the focus of this thesis and are discussed in further detail in the following section.

Evolutionary Programming Evolutionary Programming (EP; Fogel, 1962; Fogel et al., 1966) was initially developed to evolve finite state automata. However, EP adheres to no fixed representation of individuals and has since adopted real-valued representations. EP shares many characteristics with ES.

Genetic Programming Genetic Programming (GP; Koza, 1992) evolves computer programs that are represented as tree structures.

2.1.1 Evolution strategies

Evolution Strategies (ES) are well suited for application to continuous optimisation problems. They are characterised by:

- individuals represented by real-valued vectors;

- the use of mutation as the primary variation operator; and
- self-adaptation of the mutation properties during operation.

A brief overview of ES shall be given in this section. The discussion will focus on those characteristics of ES that are of most relevance to the work in this thesis. A more comprehensive introduction to ES can be found in Eiben and Smith (2003) and Beyer and Schwefel (2002).

ES were proposed by Rechenberg (1973) and Schwefel (1975) for the application of experimental parameter optimisation. One of the first applications of ES was the shape optimisation of a two-phase jet nozzle for superheated water (Klockgether and Schwefel, 1970). The nozzle was assembled from a total of 330 compatible segments with conically shaped interiors, which permitted the assembly of 10^{60} different nozzle configurations without discontinuous contours. Starting from an analytically derived nozzle shape, an unusually shaped optimal configuration was found that raised efficiency from 55 percent to nearly 80 percent. It is worth noting that this experiment was not conducted in simulation, but on an experimental apparatus in the real world on a noisy and multimodal problem. For a brief history of ES see Beyer and Schwefel (2002).

Self-adaptation of strategy parameters

An important characteristic of ES is the self-adaptation of *strategy parameters* that control the evolutionary search process (Bäck, 2000). A self-adaptive EA includes these strategy parameters in the genotype of each individual such that they evolve along with the candidate solution. Thus, the strategy parameters are themselves subject to selection and variation: not some deterministic rule. These parameters must otherwise be set in advance by the user and remain constant during execution.

Self-adaptation of the strategy parameters is advantageous for several reasons:

- Constant strategy parameters yield suboptimal performance for many problems, because the optimal value of these parameters will change as the search process progresses. For example, consider the mutation variance during the search process (Eiben and Smith, 2003). During the beginning of the search process large mutations are necessary in order to adequately explore the search space. However, as the search process progresses

smaller mutations become more appropriate as a solution is approached and only fine tuning is required.

- The fitness function can change during the search process and the EA will react appropriately. Continuing the example from above, if the fitness function were to change as a solution was being approached the mutation variance could be increased and a new phase of exploration would begin.

- Detailed knowledge about the most suitable choice of strategy parameters is typically not available and is problem dependant. Thus, self-adaptation relieves the user from the laborious process of fine tuning the strategy parameters by hand.

However, self-adaptation of the strategy parameters does introduce some problems:

- The user must set a *learning rate* that controls the rate of adaptation of the strategy parameters. Based on experimental and theoretical investigations, the suggested value of this parameter for ES is proportional to the inverse of the square root of number of object variables.

- Self-adaptation can cause *divergence* as well as *premature convergence* (Meyer-Nieberg and Beyer, 2007). For example, consider the effect of self-adapted mutation variance. Divergence will occur if the mutation variance increases too quickly and causes the object variables to grow without bound. Conversely, premature convergence will occur if the mutation variance is decreased too quickly causing the EA to inadequately explore the search space and converge on a local optimum.

- The inclusion of strategy parameters in the genotype increases the total number of variables that must be optimised by the EA, which can increase the total number of fitness function evaluations required to find a solution. This is acceptable only if the problem requires the additional complexity.

Nevertheless, it has been demonstrated that self-adaptation of the strategy parameters can increase the performance of EAs for many problems (Beyer and Deb, 2001).

Overview of the algorithm

In ES, an individual's genotype consists of two components: *object variables*, $\boldsymbol{x}$, that represent points in the search space and strategy parameters that control the statistical properties of the mutation operator. The number of object variables n is equal to the dimension of the problem. The type and number of strategy parameters depends on the variant of the mutation operator used.

The concept of including the strategy parameters that control mutation in the genotype is known as Mutative Strategy Parameter Control (MSC; Ostermeier and Hansen, 1999; Hansen and Ostermeier, 2001). The objective of MSC is to increase the probability of mutation steps that have been successful in the past based on the assumption that they will also be successful in the near future.

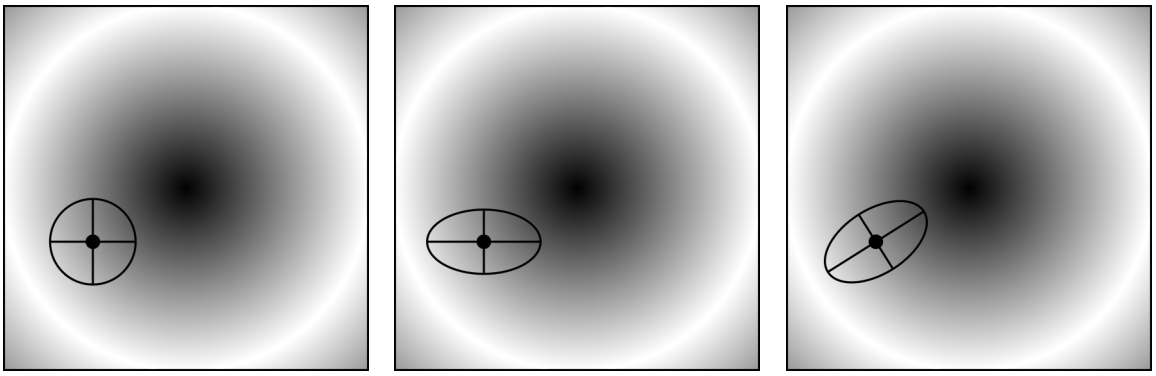
The primary variation operator in ES is mutation. The object variables of an individual are mutated by adding random values drawn from a normal distribution with a mean of zero and a shape determined by the individual's strategy parameters. Two strategy parameters may be used: the mutation step size σ which is the standard deviation of the normal distribution; and the rotation angle α which is used to calculate a covariance matrix that rotates the normal distribution. The number of strategy parameters are given by n_σ and n_α respectively.

To clarify, the *variance* of a probability distribution measures the spread of its values about the mean and is equal to the standard deviation squared. The *covariance* measures how much two random variables vary together about their respective means. The covariance matrix is a matrix of covariances between elements of a vector. *Correlation* is closely related to covariance and indicates the strength of a linear relationship between two random variables.

Eiben and Smith (2003) identify three canonical variants of the mutation operator, which are distinguished by their use of strategy parameters to define the shape of the normal distribution:

Uncorrelated mutation with one step size The genotype includes a single σ value that is used for all object variables $n_{sigma} = 1$. For the two dimensional case shown in Figure 2.2(a), lines of equal probability form circles. The probability of moving in any direction is the same irrespective of the effect on fitness.

Uncorrelated mutation with n step sizes The genotype includes a σ value for each of the



(a) Uncorrelated mutation with one step size

(b) Uncorrelated mutation with n step sizes

(c) Correlated mutation

Figure 2.2: Examples of the three canonical variants of the mutation operator used in Evolution Strategies. Lines of equal probability for a two dimensional normal distribution are shown. Regions of higher fitness are darker.

object variables $n_\sigma = n$. For the two dimensional case shown in Figure 2.2(b), lines of equal probability form ellipses. This increases the probability of moving along those dimensions with the greatest effect on fitness.

Correlated mutation The genotype includes a σ value for each object variable $n_\sigma = n$ and a α value for each combination of object variables $n_\alpha = \frac{(n^2-n)}{2}$. For the two dimensional case shown in Figure 2.2(c), lines of equal probability form rotated ellipses. This increases the probability of moving in the direction with the greatest effect on fitness. (Rudolph, 1992)

The strategy parameters are also updated using a normal distribution, although details will not be provided here.

Recombination is a secondary variation operator in ES. Two variants of recombination are used to create children:

Discrete recombination Discrete recombination randomly selects a value from the parent vectors at each position in the child vector.

Intermediate recombination Intermediate recombination calculates the arithmetic mean of the parent vectors to create a child.

Different recombination variants may be applied to the object variables and strategy parameters. More than two parents can be used in recombination where the number of parents is given by ρ.

Each generation λ children are created by a process of recombination followed by mutation. Parents are selected from the population randomly with a uniform distribution: this process is not biased by fitness.

Survivor selection in ES is a deterministic process. Survivors are ranked in order of fitness and the best μ are selected for the next generation. There are two canonical versions of survivor selection:

$(\mu/\rho, \lambda)$ **selection** The best μ are selected from the children only.

$(\mu/\rho + \lambda)$ **selection** The best μ are selected from the union of the parents and the children.

Typically, λ is much higher than μ.

2.1.2 Covariance matrix adaptation evolution strategy

The Covariance Matrix Adaptation Evolution Strategy (CMA-ES) is a *derandomised* ES that self-adapts the covariance matrix of the mutation distribution and a global mutation step size. It is derandomised because these strategy parameters are adapted by deterministic rules based on past successful mutation steps. Originally designed for small population sizes, the CMA-ES is both a robust local and global search algorithm with demonstrated competitive performance on unimodal (Hansen and Ostermeier, 2001) and multimodal test functions (Kern et al., 2004; Hansen and Kern, 2004). It is well suited to non-separable and ill-conditioned problems. This section presents a summary of the major features of the algorithm. For full implementation details see Hansen and Ostermeier (2001); Hansen and Kern (2004).

Overview of the algorithm

For each generation g, the λ children of the next generation $g+1$ are generated by:

$$\boldsymbol{x}_k^{(g+1)} \sim \boldsymbol{m}^{(g)} + \sigma^{(g)} \boldsymbol{N}\left(0, \mathbf{C}^{(g)}\right) \qquad \text{for } k = 1, \ldots, \lambda \tag{2.1}$$

where $\boldsymbol{x}$ are the object variables of an individual, σ is the global mutation step size, $\boldsymbol{m}$ is the weighted mean of the best individuals in the population and $\mathcal{N}\left(0, \mathbf{C}^{(g)}\right)$ is a multivariate normal distribution with a mean of zero and covariance matrix $\mathbf{C}$.

Before advancing to the next generation the weighted mean of the best children $\boldsymbol{m}^{(g+1)}$ must be calculated and the self-adaptive strategy parameters must be updated.

The calculation of $\boldsymbol{m}^{(g+1)}$ combines the selection of survivors, selection of parents and recombination operators into a single process. It is equivalent to selecting μ survivors from the children only and performing weighted intermediate recombination of the entire population to produce a single child. The λ children are ranked in order of fitness and the weighted mean calculated as:

$$\boldsymbol{m}^{(g+1)} = \sum_{i=1}^{\mu} w_i \boldsymbol{x}_{i:\lambda}^{(g+1)} \tag{2.2}$$

where $\boldsymbol{x}_{i:\lambda}$ is i-th best individual and w_i is a recombination weight coefficient. The recombination weight coefficients have the following properties:

$$\sum_{i=1}^{\mu} w_i = 1 \quad \text{and} \quad w_1 \geq w_2 \geq \ldots \geq w_\mu > 0$$

Self-adaptation of the strategy parameters

Self-adaptation in the CMA-ES operates on two strategy parameters: the covariance matrix $\mathbf{C}$ and the global mutation step size σ. These parameters are updated using the evolution path.

An *evolution path* is a sequence of successful mutation steps taken over consecutive generations (Hansen and Ostermeier, 1996). It reveals correlations between consecutive successful mutation steps. If the mutation steps are anti-parallel correlated then the evolution path will be short. Conversely, if the mutation steps are parallel correlated then the evolution path will be long. The length of the evolution path (short or long) is determined by comparing it against the expected length of the path under random selection. Hansen and Ostermeier (1996) argue that the most efficient mutation steps are uncorrelated, which occurs under random selection. This is because anti-parallel correlated mutation steps cancel each other out and should be shortened, while parallel correlated mutation steps are going in the same direction and should be lengthened.

The *rank-one update* of the covariance matrix uses the evolution path $\boldsymbol{p}_c$ that is calculated using exponential smoothing of the weighted mean of the mutations added to the best

individuals or $m^{(g+1)} - m^{(g)}$. Additionally, Hansen and Kern (2004) introduced the *rank-μ update* as an extension to the rank-one update that can make use of the information contained in large populations. It is calculated using exponential smoothing of the weighted sum of the differences between the best μ children and the weighted mean of the best individuals from the population or $\boldsymbol{x}^{(g+1)} - \boldsymbol{m}^{(g)}$. The rank-one and rank-$\mu$ updates are combined to update $\mathbf{C}$.

The global mutation step size is updated using the *conjugate* evolution path $\boldsymbol{p}_\sigma$. It is calculated in a similar manner to $\boldsymbol{p}_c$ except that $m^{(g+1)} - m^{(g)}$ is transformed by $\mathbf{C}^{-\frac{1}{2}}$ to make the expected length of the evolution path independent of direction.

Beyer and Arnold (2003) presented empirical evidence that the adaptation of the mutation operator may be sub-optimal for noisy fitness functions. As this result was restricted to a specific test case, it is unknown how it translates to real world problems. However, it is important that practitioners are aware of this possibility when applying the CMA-ES to problems.

External strategy parameters

The CMA-ES is quasi-parameter free (Hansen and Kern, 2004). This does not mean that there are no external strategy parameters that must be set by the user, but rather that robust defaults are included in the algorithm description. These defaults were chosen based on empirical investigations using simple test functions (Hansen and Ostermeier, 2001). So, in addition to the population size λ and parent number μ there are five parameters that are used for the self-adaptation of the covariance matrix and global step size. These parameters include normalisation and damping factors. The recombination weight coefficients w_i for $i \in [1, \mu]$ must also be defined. The default λ and μ values are given by:

$$\lambda = 4 + \lfloor 3 \ln n \rfloor \tag{2.3}$$

$$\mu = \frac{\lambda}{2} \tag{2.4}$$

For the defaults of the remaining external strategy parameters see Hansen and Kern (2004).

Generally it is recommended that the default values be used for all external strategy parameters with the exception of the population size. Increasing the population size may improve performance on some multimodal problems (Hansen and Kern, 2004).

Two problem dependent values must be set during initialisation of the CMA-ES: the initial global mutation step size $\sigma^{(0)} \in \mathbb{R}^+$ and the initial search point $\boldsymbol{m}^{(0)} \in \mathbb{R}^n$. Generally it

is recommended that $\sigma^{(0)}$ be set to half of the interval from which $\boldsymbol{m}^{(0)}$ is randomly and uniformly sampled. Otherwise, the initial evolution paths are set as $\boldsymbol{p}_c^{(0)} = 0$ and $\boldsymbol{p}_\sigma^{(0)} = 0$ and the initial covariance matrix is set as $\mathbf{C}^{(0)} = \mathbf{I}$.

2.2 Artificial neural networks

Artificial Neural Networks (ANN) are mathematical and computational models of the biological neural networks that make up the brain. From an engineering perspective, they are useful tools that can be used to model complex relationships, find patterns in data and control dynamical systems.

An ANN, or simply Neural Network (NN), is a parallel distributed processor made up of simple processing units called *neurons*. Each neuron computes an *activation potential*, which is the weighted sum of its inputs plus an additional constant term called a *bias*. The output from a neuron is calculated from the activation potential using an *activation function* that squashes the output between some range. Mathematically, a neuron k is described by the following equations (Haykin, 1999):

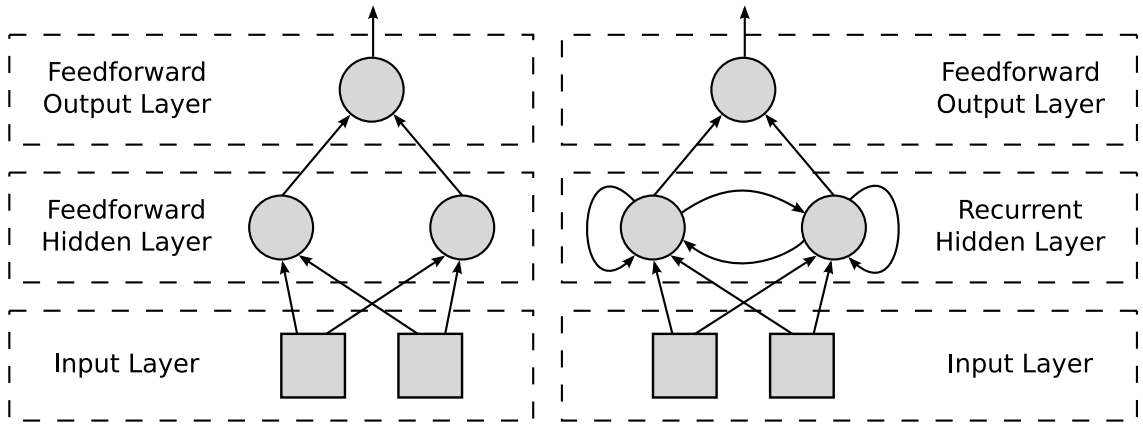
$$v_k = \sum_{j=0}^{m} w_{kj} x_j \tag{2.5}$$

$$y_k = \varphi(v_k) \tag{2.6}$$

where $x_0, x_1, \ldots, x_m$ are the input signals, $w_{k0}, w_{k1}, \ldots, w_{km}$ are the *connection weights* of neuron k, v_k is the activation potential, $\varphi(\,\cdot\,)$ is the activation function and y_k is the output signal. The activation function is typically a sigmoid function, which is s-shaped and exhibits a balance between linear and nonlinear behaviour.

The manner in which the neurons are arranged in a NN strongly governs its behaviour. Neurons are typically arranged in layers. A single-layer NN consists of an input layer of source nodes, in which no computation occurs, that is connected to an output layer of neurons. A multilayer NN inserts one or more hidden layers of neurons between the input and output layers. Generally, there are two different classes of NN architecture:

Feedforward networks In a feedforward NN connections occur between adjacent layers in the direction from the input layer to the output layer only. The NN is *fully connected*



(a) Feedforward neural network (b) Recurrent neural network

Figure 2.3: Neural network architectures. (a) Fully connected multilayer feedforward neural network. (b) Fully connected multilayer recurrent neural network with a fully recurrent hidden layer.

if every node in a layer is connected to every other node in the adjacent forward layer. Figure 2.3(a) shows an example of a fully connected multilayer feedforward NN.

Recurrent networks In a Recurrent Neural Network (RNN) there is at least one feedback or *recurrent* connection between neurons. A recurrent connection is one where the output from a neuron is fed back as an input to a neuron in the same layer or a preceding layer. Likewise, a self-recurrent connection is one where the output from a neuron is fed back as one of its inputs. For a layer to be *fully recurrent*, recurrent connections must exist between every neuron in the layer and all neurons in the layer must be self-recurrent. The introduction of recurrent connections results in the NN exhibiting dynamical behaviour. Figure 2.3(b) shows an example of a fully connected multilayer RNN with a fully recurrent hidden layer.

2.3 Neuroevolution

Neuroevolution (NE) is the application of EAs to the design of NNs. EAs are used in combination with NNs on three different levels (Yao, 1999):

Connection weights The connection weights of a fixed NN architecture are evolved using

an EA. Standard EA techniques, such as ES, can be used. The connection weights are directly encoded in the solution's object variables using either a binary or real-number representation dependant on the EA used.

Architecture An EA can be used to evolve the NN architecture, which includes the topology, transfer functions and connection weights. Standard EAs may be used, but specialised EAs have been developed specifically for the purpose of evolving NN architectures. The architecture can be either directly or indirectly encoded in the genotype. The direct encoding scheme explicitly represents every connection and neuron that will make up the NN. Conversely, the indirect encoding scheme specifies developmental rules that are used to construct the NN. Evolving the NN architecture may permit architectures to be found that are specialised for a particular problem without the intervention of a human designer.

Learning rules An EA can be used to evolve the learning rules used to train the NN. The connection weights of the NN will then be adapted using these learning rules in a separate process. Evolving the learning rules can be seen as a process of *learning to learn*.

For a comprehensive review of early work in NE see Yao (1999) and for a review of recent work see Floreano et al. (2008). This section will review recent work in NE with a specific focus on those algorithms that have been applied to Reinforcement Learning (RL) problems.

2.3.1 Evolution of connection weights

Both ES and Evolutionary Programming (EP) have been applied successfully to the evolution of connection weights for a variety of problems by a number of authors. These algorithms use a real-valued representation of the connection weights and are well suited to continuous optimisation problems. Some early work was also conducted using Genetic Algorithms (GA) with a binary representation.

Agent learning in games is one area where neuroevolution has received some interest (Lucas and Kendall, 2006). It has been applied to board games, such as checkers (Chellapilla and Fogel, 2001), and more recently computer games, such as Pac-Man (Lucas, 2005; Gallagher and Ledwich, 2007). Other applications include the control of a simulated helicopter

(De Nardi et al., 2006) and simple cars in racing games (Togelius et al., 2007). Many of these applications have evolved connection weights using simple versions of ES or EP either without self-adaptation or self-adaptation of mutation step sizes only. So, there is scope for further work using EAs that make greater use of self-adaptation.

The CMA-ES has been used to evolve connection weights for only a very limited number of applications. RL problems include only the inverted pendulum problem (Igel, 2003), although the CMA-ES has also been used for many other optimisation problems.

A few specialised EAs have been proposed for the evolution of connection weights. Symbiotic, Adaptive Neuro-Evolution (SANE; Moriarty, 1997) evolves individual neurons rather than complete networks. The neurons are combined to form the hidden layer of a two layer NN. Combinations of neurons that perform well together are maintained and these *blueprints* are also evolved. Moriarty (1997) observed that subpopulations of neurons spontaneously formed in the main population and that these subpopulations corresponded to neurons with specialised roles in the NN. The ability of SANE to evolve feed-forward NNs was demonstrated for several benchmark problems. However, SANE was unable to reliably evolve Recurrent Neural Networks (RNN). This is because the response of a neuron in a RNN is dependent on all the other neurons in the hidden layer to which it is connected, whereas the response of a neuron in a feedforward NN is not. As SANE continuously combines different neurons, the evolution of the interdependent relationships between neurons in a RNN is obstructed.

Gomez (2003) proposed Enforced SubPopulations (ESP) as an improvement to SANE. ESP also evolves individual neurons, but each neuron in the NN is assigned to a separate subpopulation. Recombination may only occur between the members of the same subpopulation. This permits the neurons in each subpopulation to quickly evolve to perform specialised roles within the assembled NN and enabled ESP to reliably evolve RNNs. Neither SANE nor ESP evolve network topology, although ESP can increase and decrease the size of the hidden layer by adding and removing subpopulations respectively.

ESP was applied by Gomez and Miikkulainen (2003) to the stabilisation of a finless rocket. Although no external disturbances were included in the finless rocket experiments, Gomez and Miikkulainen (2004) demonstrated the robustness of ESP in the presence of external disturbances on the inverted pendulum problem. Also, Sit and Miikkulainen (2005) applied ESP to the control of a simplified, two dimensional model of a small robot designed to assist astro-

nauts in space.

Cooperative Synapse Neuroevolution (CoSyNE; Gomez et al., 2006, 2008) is similar to ESP in that it evolves several separate subpopulations, but each subpopulation is associated with NN connection weights rather than neurons. One subpopulation exists for each feedforward and recurrent connection between neurons within a predefined NN topology. The subpopulation size determines the number of NNs that are created and evaluated each generation. To create a NN, connection weights are selected from each subpopulation and assigned to their associated connection within the NN. Each generation children are generated from the parents using crossover and mutation. Additionally, the subpopulations are randomly permutated so that connection weights potentially form part of a different NN in the next generation.

2.3.2 Evolution of architecture

Several specialised EAs have been proposed for the evolution of architecture. Three recent methods that have been applied to RL problems are described in this section. In particular, all of these methods have been applied to the inverted pendulum problem studied in Chapter 5. For a taxonomy of approaches to the evolution of architecture see Stanley and Miikkulainen (2003).

NeuroEvolution of Augmenting Topologies (NEAT; Stanley, 2004) evolves the topology and connection weights of a NN from a minimal starting point. Each genotype contains a list of connection genes and node genes. Mutation can change connection weights by adding a random real number from a uniform distribution. Structural mutations can add new connection genes and new node genes. NEAT uses *innovation numbers*, which are assigned to new genes, to represent the chronology of genes in the population. These innovation numbers permit crossover to occur between NNs with different topologies because it is possible to identify genes that represent the same structural features. Furthermore, innovation numbers permit the population to be divided into species based on topological features. Organisms within a species primarily compete with other organisms in the same species rather than with the entire population. This helps to prevent organisms with topological innovations (new neurons or connections), which usually decrease fitness, from being lost from the population. A real time version of NEAT has been developed that permits the evolution of computer game agents as

the game is being played (Stanley et al., 2005).

Evolutionary Acquisition of Neural Topologies (EANT; Kassahun, 2006) also evolves complete networks. The *linear genome* (genotype) is interpreted as a tree based program where the topology of the NN is implicitly represented by the order of the genes. The genes can be neurons, inputs to the NN or *jumpers* (connections) between neurons. EANT operates over two evolutionary time scales. At the shorter time scale the connection weights are mutated using the uncorrelated mutation operator from ES, while at the longer time scale the topology is mutated by adding new neurons and adding or removing jumper connections. Crossover is performed in a manner similar to NEAT. EANT2 (Siebel et al., 2007) is an improved version of EANT that replaces the ES used to mutate connection weights with a CMA-ES.

Analog Genetic Encoding (AGE; Mattiussi, 2005) is an implicit NE method inspired by Gene Regulatory Networks (GRN). NNs are encoded in the genotype as sequences of characters from a finite alphabet with neuron sequences separated by predefined delimiter sequences. Connection weights are determined by an interaction map, which is a mathematical function that calculates a numerical value from two sequences of characters. A NN is constructed by scanning the genotype for the sequences of characters representing neurons and computing connection weights from the interaction map.

The methods described above all propose different competing solutions to the same problem. That is, how to incrementally evolve NN architectures by adding neurons and connection weights. Such an approach must be followed, if the search space is to be constrained to a reasonable size. These methods are all motivated by the assumption that evolving NN architectures can result in solutions that could not be as easily found by designing the architecture in advance and evolving the connection weights.

2.4 Learning and evolution

In biological organisms evolution and learning are two fundamental forms of adaptation that occur over different time frames (Nolfi and Floreano, 1999). Evolution occurs over several generations of organisms and permits a species to adapt to long-term environmental changes. Conversely, learning occurs within the lifetime of a single organism and permits the organism to adapt to short-term environmental changes.

In the field of ANNs, learning is the process through which the connection weights of a NN are calculated. It is defined as "a process by which the free parameters of a neural network are adapted through a process of simulation by the environment in which the network is embedded." (Haykin, 1999, p. 50) Thus, the connection weights represent stored knowledge which is defined as information used to "interpret, predict, and appropriately respond"(Haykin, 1999, p. 23) to the environment.

Three major learning paradigms model the environment in which learning occurs:

Supervised learning In supervised learning, an agent learns by emulating a *teacher*. The teacher has knowledge of the environment represented by input-output examples known as the *training set*. These examples are presented to the agent, which is adjusted based on the error between the output of the teacher and the output of the agent for a particular input. The goal is to learn the input-output mapping of the environment such that the agent can provide the correct output when given some input once the teacher is removed. Examples of supervised learning include pattern recognition and function approximation.

Unsupervised learning In unsupervised learning, an agent learns by discovering patterns present in the environment. The agent receives only inputs, because no teacher is present to provide target outputs and no rewards are received from the environment. Instead, the agent is updated according to rules that specify what aspects of the inputs should be captured in the outputs. The goal is to learn an input-output mapping that captures the desired characteristics of the environment. Clustering is an example of unsupervised learning.

Reinforcement learning In Reinforcement Learning (RL), an agent learns by interacting with its environment in a sequence of discrete steps. At each step the agent senses the state of the environment and takes some action. As a result of this action the state of the environment changes and the agent receives some reward that measures the desirability of this new state. The goal is to learn a mapping of states to actions, called a *policy*, that maximises the long term reward received by an agent. Examples of RL include robot control and game playing.

Note that while this thesis is primarily concerned with Reinforcement Learning (RL) problems, it does not apply a RL technique.

Generalisation refers to the ability of a NN to produce reasonable outputs for inputs that were not present in the training set. This is an important concept because it is hoped that generalisation reduces the amount of training required by a NN to learn optimal policies.

A NN is said to be *overfitted* if it matches the training set very accurately, but generalises poorly. In this case the NN has learnt a feature that is present in the training set, but is not true of the underlying function to be modeled.

2.4.1 Evolutionary learning

Even though NE was inspired by biological evolution, it is typically applied to learning problems in a manner more similar to that of a learning algorithm. As learning is defined as a process that adapts the free parameters of a NN, the evolution of connection weights or topology with connection weights emulates a learning algorithm. This process can, therefore, be described as *evolutionary learning* and, as such, the terms *learn* and *evolve* can be used interchangeably.

NE begins to distinguish itself from other learning algorithms when it is used to evolve topology without connection weights, which are learnt in an independent process, and when it is used to evolve learning rules.

NE does not fit neatly into any of the three major learning paradigms. However, it can be applied to problems normally associated with all three paradigms if a suitable fitness function can be defined. For example, NE can be used for supervised learning problems by setting the fitness of a solution proportional to the error between the output of the evolved NN and that of the teacher.

2.4.2 Comparison between evolutionary and reinforcement learning

A brief comparison between NE and RL shall be presented before continuing. This comparison will permit greater insight into the advantages and disadvantages of NE as a technique for the problems considered in this thesis. Furthermore, there appears to be confusion in some previously published work with respect to the relationship between NE and RL.

NE and RL share many common characteristics. They both involve learning while interacting with the environment and are applied to the same class of problems where a policy consisting of a sequence of actions must be found that maximises some long-term reward. Such problems are typically described as being RL problems. However, despite the apparent similarities between NE and RL they are different and, while NE may be applied to RL problems, it is not a RL method.

Sutton and Barto (1998) discussed the differences between RL and evolutionary methods in terms of *trial-and-error learning*. Consider the major elements of trial-and-error learning:

Selectional Try alternative actions and select among them by comparing consequences.

Associative Associate actions found by selection with particular situations.

Evolutionary methods are selectional, but not associative. They select entire policies based on a fitness score, which is a singular scalar value. The states experienced and actions taken by a solution are not considered beyond the calculation of the fitness score. So, specific actions cannot be associated with specific states. Conversely, RL is both selectional and associative. It maps each state or state-action pair to an immediate reward and assigns values based on the expected rewards from states that follow. Actions are then selected to maximise the expected future reward. For these reasons Sutton and Barto (1998) concluded that evolutionary methods are not well suited to RL problems where state information is available. Nevertheless, NE has been applied to RL problems with some success.

A number of comprehensive comparisons between NE and RL have been published using the inverted pendulum problem (Whitley et al., 1993; Moriarty and Mikkulainen, 1996; Gomez et al., 2008). These studies show that NE methods can find solutions in this problem domain with fewer function evaluations than RL methods. In particular, NE is shown to significantly outperform RL for variations of the inverted pendulum problem that contain hidden state. The inverted pendulum problem is discussed further in Chapter 5.

A limited number of direct comparisons between NE and RL have been published in other domains. These studies have commonly used some form of Temporal Difference (TD) learning, which represents one of the fundamental classes of RL methods Sutton and Barto (1998). Taylor et al. (2006) compared NEAT with the TD method Sarsa on the keepaway robot soccer task. The results showed that NEAT was able to find better policies than Sarsa, but required

more evaluations to do so. Lucas and Kendall (2006) provides an overview of comparisons conducted by several authors using games, which will not be discussed in detail here.

Overall, no firm conclusions can be drawn regarding the relative performance of NE and RL due to the limited number of problem domains studied. However, the results do suggest that for some problems NE methods can produce better solutions than RL. Future comparisons are not required to determine which method provides the best performance, but rather the properties of the problems for which each method is best suited. This will provide a greater understanding of the strengths and weaknesses of these approaches.

2.5 Evolutionary robotics

Evolutionary Robotics (ER) is a technique for the synthesis of autonomous mobile robots via an automatic design process involving EC. It follows the general method of an EA (see Section 2.1) with the following characteristics:

- Each genotype encodes the control system and/or morphology of a robot.
- Each genotype is evaluated by instantiating it as a robot in a simulated or physical environment. The robot is permitted to interact with its environment (move, sense, manipulate) as dictated by its control system and morphology.
- The fitness function measures the performance of the robot at some task in the environment. For example, tasks may include driving in a straight line and avoiding collisions with obstacles.

Any form of EC can be used in ER, so long as the robot control system or morphology can be represented in the genotype. ER shares many characteristics with techniques such as behaviour-based robotics and artificial-life. For a complete introduction to ER see Nolfi and Floreano (2000).

The appeal of ER is highlighted by two possibilities. Firstly, solutions to the complex problem of robot design may be found via a automatic design process. Secondly, that novel solutions may be found via the exploration of regions in the design space beyond the constraints of conventional techniques and the imagination of human designers.

The majority of the work conducted in ER has focused on the evolution of robot controllers, which have typically been implemented using NNs. The neurocontrollers have been evolved using either GAs, ES or biologically inspired EAs specifically designed for the evolution of robot controllers. For these reasons the fields of ER and NE are closely related.

2.5.1 Philosophy

ER is a bottom-up approach to the design of robot control systems. It is based on the following principles:

Situated The robot is able to sense and interact with the world.

Embodied The robot exists as a physical entity in the world.

Emergence The behaviour of the robot results from the dynamic interaction between the robot's control system, body and the environment.

These principles are shared with Behaviour-Based Robotics (BBR; Arkin, 1998).

In BBR a robot control system is decomposed into modules designed to achieve some task and the overall behaviour of the robot emerges from the interaction between the individual modules. The Subsumption Architecture (Brooks, 1986) was the first approach proposed to BBR. In the Subsumption Architecture, simple behavioural modules are arranged in hierarchical layers with higher levels responsible for increasingly abstract behaviours. For example, obstacle avoidance is a low level behaviour and explore environment a high level one (Brooks, 1986). All layers have access to sensory information and the higher level layers may inhibit or suppress signals from lower levels.

Proponents of ER have argued that it is difficult for a human designer to break a complex problem up into smaller problems and predict the emergent behaviour of the solution (Harvey et al., 1993). Also, superior solutions may be excluded by a human designer because they are unable to perceive them. The solution proposed by ER to these problems is that a process of artificial evolution be used to evolve robot controllers instead. Assumptions about what modules are required to achieve some desired behaviour are no longer required, so long as the behaviour is either directly or indirectly included in the fitness function (Harvey et al., 1993).

In ER, a robot's sensors are directly connected to its actuators via an evolved *brain*. The brain is typically a NN, but may also be a program evolved using Genetic Programming (GP). Little or no pre-processing is performed on the sensor information. The argument is that sensorimotor coordination allows "pure reactive agents to solve problems that, from an external perspective, apparently require more complex solutions that rely on internal states or internal representations of the external environment." (Nolfi, 2002, p. 144) However, the agents need not be purely reactive and some internal state indirectly encoding information about the environment may be necessary in some cases.

2.5.2 Applications

Generally, experiments in the field of ER have been limited to simple robots in simple environments. The most widely used robotic platform is the Khepera miniature robot (Mondada et al., 1994). The advantage of the Khepera is that it is small, inexpensive, reliable and easy to model due to its clean simple design. However, the Khepera is not representative of real robotic platforms because of its limited mobility and small size that restrict its application to desktop environments in the laboratory (Adams, 2006). As the Khepera can be modeled very accurately, the controllers evolved in simulation can be easily transferred to the physical platform. This is an advantage when conducting experiments in the laboratory to demonstrate a concept, but does not reflect the reality of most robotic platforms where there is often a high degree of uncertainty in the model of the vehicle dynamics, sensors and actuators. Furthermore, the Khepera operates in desktop environments that are considerably simpler than real robot environments, which are inherently dynamic and unpredictable (Thrun et al., 2005).

Controllers have been evolved for the Khepera for a variety of simple tasks, such as wall-following and obstacle-avoidance (Harvey et al., 1993; Nolfi et al., 1994; Floreano and Mondada, 1994), light-seeking (Watson et al., 1999) and peg-pushing (Kondo et al., 1999). More complex tasks include: homing navigation to facilitate battery recharge (Floreano and Mondada, 1996), area cleaning using a gripper (Nolfi, 1997), learning in a light-switching problem (Floreano and Urzelai, 2000), navigation with a landmark (Tuci et al., 2002) and multi-robot teams (Quinn, 2000; Baldassarre et al., 2003). These proof-of-concept experiments have explored issues in ER related to the evolutionary process and the capacity of sensorimotor

neurocontrollers to express different autonomous behaviours. However, there is a significant complexity gap between these experiments and the practical application of ER.

Experiments conducted using more sophisticated robotic platforms than the Khepera are few in number.

Hornby et al. (2000b) evolved gaits for a Sony AIBO robot dog on the physical hardware. The Sony AIBO is a quadruped robot with eighteen degrees of freedom and the gaits were described by twenty real-valued parameters in a hand-designed locomotion module. Also, a ball-chasing neurocontroller was evolved in simulation and transferred successfully to the physical robot (Hornby et al., 2000a). The neurocontroller received pre-processed sensor data and output high-level control commands.

Zufferey (2005) evolved a neurocontroller to steer an indoor blimp around a square arena using visual information. The walls of the arena were covered with randomly arranged black and white vertical strips in order to ensure the presence of sufficient visual contrast. Information from the visual sensor was pre-processed to provide the neurocontroller with the contrast rate over four evenly distributed regions. The neurocontroller also received sensor information from a yaw gyroscope. The best neurocontrollers evolved in simulation were tested on a physical robot without further evolution. The evolved neurocontroller successfully stabilised the heading of the blimp while avoiding colliding with the arena walls. Attempts to apply the same method to a ultra-light indoor airplane were unsuccessful.

Competitive co-evolution was used by Nelson and Grant (2006) to evolve neurocontrollers for small robots to play a version of the competitive team game *Capture the Flag*. In this game, robots from two teams compete to find the stationary goal of the opposing team in a maze of varying configurations. The neurocontroller was provided with range data extracted from a colour camera, using a method that relied upon certain fixed geometric features of the environment. With 150 inputs this represents the most complex use of sensor information to date in ER. Although this work demonstrated that the evolved neurocontrollers were capable of navigating the maze and finding the goal of the opposing team, no evidence of cooperative behaviour was presented. The evolved neurocontrollers were able to win a "modest" majority of games in a tournament with knowledge-based controllers.

Neural-behavioural controllers were evolved by Adams (2006) using NEAT for the iRobot Roomba, a commercially available household robot for cleaning floors. The controllers con-

sisted of hand-designed augmented finite state machine modules that controlled the robot, but received inputs from the sensors through an evolved weighted network. The evolved neural-behavioural controllers showed a significant performance improvement over evolved sensori-motor neurocontrollers.

Unmanned Aerial Vehicle (UAV) navigation controllers were evolved in simulation by Oh and Barlow (2004) using multi-objective GP. The task required the UAV to locate a radar source, navigate to it using on-board sensors and then circle closely around it. Radar signals were input to the navigation controller, which output the desired roll angle of the UAV. Low-level flight control was executed by the onboard autopilot. The evolved controllers were tested on a small, wheeled robot equipped with a passive sonar system that provided the angle and amplitude of sound signals from a stationary speaker (Barlow et al., 2005).

Despite over a decade having passed since the initial proof-of-concept experiments the practical applications of ER are very few in number. This stands in stark contrast to BBR, which has seen much wider application. The most complex problems solved by ER have used a combination of simple reactive behaviours and seem easily solvable by other techniques. Furthermore, pure sensorimotor controllers have not been used for the few experiments conducted using more capable robotic platforms equipped with sophisticated sensors. The sensor data has typically been preprocessed in some way. This appears to be a practical compromise necessitated by the difficulty of evolving a controller to process the sensor data into a form that can be used, while at the same time evolving the controller to complete some task.

A striking observation from the work of Nelson and Grant (2006) and Adams (2006) is the poor performance of the evolved controllers relative to manually designed controllers. Nelson and Grant (2006) do observe a small performance gain, but acknowledge that the manually designed controller was "very likely" not the best possible controller (Nelson, 2003). So, it has been demonstrated that evolved controllers can approach, but not yet exceed, the performance of manually designed ones.

In short, it appears that ER fails to scale with increasing problem complexity.

2.5.3 Challenges

Matarić and Cliff (1996) conducted a survey of the challenges in ER, which remains relevant

today over ten years later. Recently, Adams (2006) revisited those issues related to evolution on physical robots and in simulation.

Evolution on real hardware versus in simulation

In robotics, and engineering in general, there is a requirement to develop and test complex systems in simulation before deployment in the real world. However, the interaction of robots with the real world is difficult to accurately simulate due to noisy sensors, noisy actuators and complex dynamical interactions with other objects. This is particularly relevant to ER, because there is a real danger that controllers evolved in simulation will not transfer to the real robot. For this reason, the issue of evolution on real hardware versus evolution in simulation has received much attention in the ER literature.

From an engineering perspective, evolution on real hardware is impractical for the vast majority of robotic platforms that are more complex than the Khepera. First and foremost, the use of real hardware entails a significant cost that cannot be ignored. In addition to the cost of building or purchasing one or more robots, other costs must be considered such as the maintenance costs associated with the physical wear caused to the robot's systems by many thousands of fitness function evaluations. Secondly, many of the controllers generated during the evolutionary process will be unsafe and could cause damage to the robot or cause the robot to damage the environment in which it is housed. This issue becomes especially relevant as the size and operating speed of the robot increases. Thirdly, continuous operation of the robot requires some method to recharge the onboard batteries or provide power via a tether and the robot must be able to reset its position between fitness evaluations. This is less of an issue for ground robots that can return to a battery recharge station and are operated in environments where they are less inclined to become stuck or tip over. Lastly, if the physical environment remains static during the evolutionary process then the evolved robot control systems will take advantage of specific environmental features that do not exist in the general case.

So, the only practical option is for evolution to occur completely in simulation. In addition to modelling the robot and environment, the simulation must also model the noise present in the actuators and sensors (Jacobi et al., 1995). Any abstraction made in the simulation will be exploited by the EA and result in behaviour that will not transfer to the real world. Therefore, the creation of a suitable simulation can be both a challenging and time consuming

task that increases in difficulty with the complexity of robot and its environment. Furthermore, the simulation will never model the real world with one-hundred percent accuracy and some error will always exist between the two. Nevertheless, transference of controllers evolved in simulation to the real world has been demonstrated.

The evolved controller must be robust to the discrepancies between the simulation and the real world. As long as these discrepancies are not too large, they can be considered as another source of noise. Evolutionary fine tuning on the real hardware is also impractical for most robotic platforms. However, the evolution of controllers in simulation that are capable of adaptive behaviour when transferred to the real world may be beneficial (Urzelai and Floreano, 2001).

Issues of simulation accuracy can be most effectively dealt with by isolating the evolved controller from the worst of the noise present in the real world, thereby relieving the user from simulating these effects. Thus, the controller operates on sensor data that has been pre-processed, rather than operating on raw data direct from the sensors. Similarly, the controller commands a low-level vehicular controller, rather than directly controlling the actuators. Techniques with known robustness properties can be used to design these interfaces. The simulation, therefore, need only model the world at the level seen by the evolved controller and successful transfer to the physical robot is more probable. This is the approach followed in this thesis.

Fitness function design

The design of an appropriate fitness function is a critical step in the successful application of any evolutionary method. In ER, the fitness function must measure the ability of a robot controller to undertake some task, which may be composed of several sub-tasks. This can be a challenging problem that requires some degree of understanding of the task domain and how it can be solved. This fact is often overlooked or trivialised in the ER literature, although recently it has received some attention (Nelson, 2003; Nelson and Grant, 2006). The general issues discussed here also apply to the use of NE for RL problems.

The design of the fitness function must ensure that individuals which can only partially solve the problem receive some appropriate reward irrespective of success or failure. Otherwise the randomly generated individuals in the initial population, which are unlikely to succeed

at solving the problem, will all receive the same low fitness and there will be no diversity of fitness values in the population. EAs require fitness diversity in order to function, because it enables the individuals to be ranked in order of quality so that the fittest can be advanced to the next generation. Without fitness diversity, selection of individuals for the next generation is effectively random and the evolutionary process will fail. This is commonly referred to as the *bootstrap problem*.

For difficult problems, it is often necessary to structure the evolutionary process such that the difficulty of the problem gradually increases as the individuals in the population become more capable. As the problem difficulty is constrained to the proficiency of the population, the bootstrap problem can be overcome. Incremental increase of the problem difficulty can be achieved by *scaffolding* the problem. The scaffolding is a temporary framework used to support individuals during evaluation that is gradually removed during the evolutionary process. Scaffolding can be applied at three levels:

Fitness function Additional terms can be included in the fitness function to reward partial successes or particular behaviours. This creates the problem of how to weight and combine the individual components.

Task A complex task can be decomposed into individual sub-tasks. As individuals become capable of solving a sub-task the next sub-task is introduced. If each sub-task is distinctly separate, then the behaviours learnt for previous sub-tasks may be lost when learning the current sub-task. Scaffolding of the task is also known as *incremental evolution* or *shaping*.

Environment and agent The environment and agent can be simplified, modified or supplemented with additional information.

Scaffolding at one or more of these levels may be necessary.

For example, consider the problem of evolving a controller for an indoor aircraft in simulation. If the aircraft is unstable, it is unlikely that a randomly generated controller will be able to fly the aircraft for very long before crashing. Scaffolding may be included to increase the stability of the aircraft, or the controller may responsible for a single degree of freedom while a human designed controller is responsible for the others. A similar approach was followed by De Nardi et al. (2006) for the control of a simulated miniature helicopter.

The use of scaffolding raises issues related to the explicit-implicit characteristics of the fitness space (Floreano and Urzelai, 2000). An explicit fitness space contains significant human bias introduced by scaffolding and is typical of conventional optimisation problems. Conversely, an implicit fitness space is largely free from human bias and permits the emergence of truly novel solutions. In practice, however, implicit fitness spaces are inevitably plagued by the bootstrap problem and most experiments in ER have used some form of scaffolding. Recall that scaffolding may be applied at the environment and agent level, so a user may unintentionally scaffold a problem through the specification of these experimental parameters. This criticism applies to most of the experiments in ER that have been conducted in simple desktop environments with simple robots and may be one of the reasons why ER has failed to scale up with increased complexity.

Co-evolution

In the normal evolutionary process the fitness of an individual is evaluated independently of all other individuals in the population. Alternatively, in a *co-evolutionary process* the fitness of an individual is dependant on its interaction with other individuals from the same population or a separate population. Interactions in the co-evolutionary process can be either cooperative or competitive. Cooperative co-evolution is well suited to the evolution of distributed systems, such as multi-agent systems (Panait and Luke, 2005). Conversely, competitive co-evolution is well suited to games that can be defined in terms of competitive success between agents. A common application of competitive co-evolution in ER is the predator-prey problem where one individual, the predator, receives fitness for capturing another individual, the prey (Cliff and Miller, 1995; Nolfi and Floreano, 1998).

The promise of co-evolution is that it may incrementally and continuously increase the problem difficulty without the need for scaffolding. In competitive co-evolution this is known as an *arms race*, where progress is made by adaptations and counter-adaptations against competing individuals. However, co-evolution is no silver-bullet and it does not guarantee a continuous increase in complexity. In particular, the co-evolutionary process is prone to a dynamical effect where the population cycles through the same set of possible adaptations, each exploiting the weaknesses of the previous one in turn across generations, without any overall increase in fitness (Cliff and Miller, 1995).

2.6 Conclusions

Currently, attention in the field of ER is focused on the extension of the approach beyond simple proof-of-concept experiments. As demonstrated by recent publications in the field, this has proven to be a particularly challenging task. Application of ER to more sophisticated robots operating in unstructured environments requires the evolution of neurocontrollers that are capable of expressing complex behaviours consisting of multiple coordinated sub-behaviours. For behaviours more complex than simple reflexes, the evolved neurocontrollers must also be able to synthesise and retain long term information and to exploit this information for planning purposes. Furthermore, the evolved neurocontrollers must be integrated with sensors of greater complexity and sophistication than those used so far. This represents an increase of at least one or two orders of magnitude in the amount of sensor data that must be processed.

The motivation at the outset of this thesis was to apply the ER approach to the design of neurocontrollers for small low-cost Autonomous Underwater Vehicles (AUV). However, this proved to be a wildly optimistic and altogether unrealistic goal for reasons that are discussed in the following chapter describing early work conducted in the field. The lessons learnt from this early work shifted the focus of the thesis from applying the ER approach to studying the practical, engineering issues associated with the application of NE to RL problems. Such issues have received relatively little attention in the field of NE where the focus tends to be more on the development of new algorithms, particularly those that evolve NN architectures.

CHAPTER 3

Lessons learnt from early work

In this chapter, early work conducted in the fields of neuroevolution and evolutionary robotics is briefly discussed. This discussion provides additional context for the approach presented in Chapter 4 and may potentially prove useful to other researchers undertaking work in these areas.

3.1 Design of a new method for neuroevolution

Initially an attempt was made to follow a similar approach to that of Stanley (2004) and design a new method for NE that could evolve NN architectures and weights. Inspiration was sought from a wide variety of biological processes. However, little attention was given to the actual optimisation process performed by the method beyond those general principles underlying EC.

A major source of inspiration for this early work was work on Gene Regulatory Networks (GRN) by Reil (1999) and Geard and Wiles (2003). In biological organisms GRNs regulate the expression of genes during a developmental process (Davidson, 2006). They specify the spatial and temporal patterns in which specific genes are expressed. So, in principle similar regulatory networks could be used to assemble NNs. For a summary of approaches to NE inspired by GRNs see Stanley and Miikkulainen (2003).

Work in this area was discontinued for several reasons. Firstly, the system would likely include several dynamically interacting modules that would make it difficult to tune the behaviour of the algorithm and identify the root cause of experimental failures. Secondly, the algorithm could not be tested on simple numerical optimisation problems as is the norm for standard EAs. And lastly, similar approaches had met with limited success. So, it was decided that it would be prudent to use well studied EAs to evolve connection weights for NNs. Once

the limits of what could be achieved using this more cautious approach had been found, existing NE algorithms that evolve architecture could be investigated or the work on the design of a new NE algorithm could be continued. However, it was found that the evolution of connection weights combined with an appropriate choice of EA was sufficient to solve the problems studied in this thesis. Also, it was observed that other factors such as poorly designed fitness functions, incorrect formulations of NN inputs and outputs, and errors in the simulation environment were a more common cause of failure than was the EA. Thus, a study of these less exotic engineering issues and their influence on EAs could be beneficial.

3.2 Evolutionary robotics simulation of AUVs

Considerable effort was expended in an attempt to develop a realistic simulation of a small, low-cost AUV and its environment. The motivation behind this work was to evolve controllers following the ER approach. The sensors would be connected to the actuators through the neurocontroller. The sensors consisted of an Inertial Measurement Unit (IMU) and a forward-looking sonar and the actuators consisted of the control surfaces and rear thruster.

The simulation environment consisted of three components. The first component was a six degree of freedom nonlinear model of the vehicle dynamics that included hydrodynamic effects of the body and control surfaces and a nonlinear model of the thruster dynamics. This model was based on the work of Prestero (2001) for the REMUS AUV and implemented in Simulink. The second component integrated the vehicle model with a rigid body dynamics simulation environment named ODESSA (Wharington, 2003). ODESSA is an extension to the open source rigid body dynamics library Open Dynamics Engine (ODE). Integration with ODESSA permitted the simulation of multiple vehicles in a virtual environment that included terrain, obstacles and collisions between all bodies. The third component was a sonar simulation that was implemented in ODESSA using ray tracing. The sonar simulation was capable of handling multiple reflected returns from sonar pings and shadows behind objects on the seabed. Overall, the simulation software was a capable, yet complicated, tool for the simulation of AUVs.

Experiments using this simulation environment were discontinued due to the practical consdierations discussed below. However, the simulation environment developed for this work

is not without application, because such simulations are vital for the development and testing of AUVs (Brutzman, 1994). A controller evolved using a high-level model can be tested in conjunction with the other control systems on a simulated vehicle in a simulated environment before deployment in the real world.

3.3 Practical considerations and a change of focus

Several practical considerations led to the discontinuation of experiments using the realistic simulation environment. Firstly, it was realised that the ER method as applied to robots such as the Khepera was not well suited to autonomous vehicles. The dynamics of an underwater vehicle are "...highly nonlinear, coupled and time varying" (Yuh, 2000, p. 10) and uncertainties almost always exist in the dynamic model. Stability and safety are also major concerns even for low cost, small vehicles. These issues do not apply to the Khepera miniature robot nor generally to ground robots. Some of these issues do apply to the aerial robots studied by Zufferey (2005) and contributed to the difficulties encountered in that work. So, it was doubtful that a controller could be evolved in the full simulation environment that would be capable of safely and stably controlling the vehicle while utilising sonar information in order to perform some useful task. Also, given that fine-tuning of controllers on the real vehicle using evolution was impractical there were concerns that the controllers evolved in simulation would not transfer to the real world. Secondly, the computational resources required to run the full simulation environment were so high that the time required to perform an experiment would have been unrealistic. Lastly, experiments conducted in ODESSA for the inverted pendulum problem had identified computational stability issues that were being exploited by the EA. Given that the full simulation environment for the AUV was considerably more complex than that used for the inverted pendulum problem there were concerns that computational stability issues would be a major problem.

The original goal of applying the ER approach of evolving sensorimotor controllers that connect sensors directly to the actuators through a NN *brain* was set aside. Control of the vehicle dynamics can be more easily accomplished with a dedicated low-level vehicular controller designed using modern control theory and tested for stability and robustness. This is not meant to imply that a neurocontroller could not be used as a low-level vehicular controller, nor

that EC could not be used to find such a controller. Rather, that it is necessary to use a hybrid architecture where the low-level vehicular controller is commanded by a higher-level module. In which case, the method used to design the low-level vehicular controller becomes irrelevant. Although it may be possible to evolve an all-in-one controller by incrementally adding additional competencies, it is difficult to justify such an all encompassing approach for AUVs where the vehicle dynamics are so complex. Furthermore, none of the existing NE methods, including those that evolve NN architectures, have demonstrated the capability to solve such complex problems.

The focus of this thesis was shifted to the application of NE to control problems within the reactive or behaviour-based layer of the control architecture. This does not represent a complete rejection of the principles of ER, but rather a refinement necessitated by the control requirements of AUVs. If it is assumed that the evolved controller will command a suitable low-level vehicular controller, then a low-level model of the vehicle dynamics is not required in order to evolve interesting behaviours. This led to the work described in Chapter 6 where an AUV equipped with an appropriate low-level vehicular controller is modeled using a high-level kinematic model of the Dubins car.

3.4 Conclusions

Several lessons were learnt from this early work:

- The ER approach as applied to laboratory robots is not well suited to autonomous vehicles. However, some of the principles of ER can be applied to control problems within the reactive or behaviour-based layer of a hybrid control architecture.

- The vehicle and environment should be simulated at the highest level at which the relevant system behaviours are still represented. Complexity can be incrementally introduced into the simulation as it is demonstrated that a neurocontroller can be successfully evolved. A realistic simulation environment can then be used for testing the evolved neurocontroller before deployment on the real vehicle. This is by no means a new idea, but it is one that is easily overlooked by those who have not learnt it the hard way.

- An attempt should be made to use well studied EAs before designing new algorithms

based on biological inspiration. Just because some process occurs in nature does not mean that it will translate well to an efficient optimisation algorithm. Developing yet another NE algorithm without addressing the fundamental issues facing all such techniques is unlikely to significantly advance the field.

These lessons helped shape the approach to NE followed in this thesis and the experiments that were chosen for study.

CHAPTER 4

Approach

In this chapter, the general approach followed in this thesis to the design of neural networks using evolutionary computation for reinforcement learning problems is presented. The rationale behind the choice of an evolutionary algorithm is described followed by a discussion of pertinent implementation issues. In particular, the cause and effects of evaluation noise in the fitness function are discussed and methods to reduce its influence are presented.

4.1 The choice between evolution of architecture and connection weights

The first choice faced when approaching any evolutionary learning problem is between the evolution of NN architectures and the evolution of NN connection weights. Essentially, this is a choice between the use of an EA specifically designed to evolve NN architecture or an EA designed for numerical optimisation. Both approaches have their own advantages and disadvantages.

If connection weights are to be evolved then the architecture must be designed in advance by the user using heuristics and knowledge of the problem. This can be a difficult task, because the best architecture for any particular problem is generally unknown. The evolution of the architecture relieves the user from this task, but comes at the cost of an increase in the size of the search space that may result in a significant increase in the number of fitness function evaluations required by the EA to find a solution. Irrespective of the methods used to control complexity in EAs that evolve architecture, there must always be some increase in the size of the search space. Furthermore, EAs that evolve architecture tend to be more complex than

those that evolve connection weights. This is due to the requirement to operate on what are effectively directed graphs rather than numerical arrays. The question is then whether the increase in complexity associated with the evolution of architecture is justified for the problems under consideration.

The architecture of a NN determines the nonlinear functions that it can express. The architecture is defined by the number and size of the hidden layers, the connectivity between those layers and the properties of the neurons such as bias connections and transfer functions. Also, RNNs capable of short-term memory require the definition of self-recurrent connections and recurrent connections between neurons in the layers. When evolving connection weights, if the user chooses a NN that is too large then the size of the search space may prevent the EA from finding a solution, whereas if the NN is too small then the problem may be unsolvable. Furthermore, the problem may also be unsolvable if a specialised architecture is required of which the user is unaware. Thus, the appeal of evolving architecture over connection weights.

These issues are illustrated by the published results of various NE algorithms for the inverted pendulum problem that is studied in Chapter 5. This problem can be solved by fully connected NNs with only a few hidden neurons. Neuron bias connections are unnecessary for this problem and their inclusion doubles the number of fitness function evaluations required by the CMA-ES to evolve the connection weights of a neurocontroller (Igel, 2003). However, the CMA-ES still outperforms all NE algorithms that evolve architecture even when far more hidden neurons are used than required [1]. The cost of evolving NN architecture is demonstrated by the difference in performance between EANT and CMA-ES. EANT features a nested ES and incrementally adds neurons and connections. It required over twice the number of fitness function evaluations as the CMA-ES to find a recurrent neurocontroller of similar size (Kassahun, 2006).

The approach followed in this thesis uses a numerical EA to evolve connection weights only. NE is viewed as an optimisation process that is performed on the connection weights of a NN. Although the benefits promised by the evolution of architecture are acknowledged, it is argued that this approach remains applicable for the following reasons. Firstly, for many robotic control problems that evolutionary learning is capable of solving only small, fully connected

[1] Although CMA-ES is outperformed by CoSyNE on the inverted pendulum problem, CoSyNE does not evolve NN architecture.

NNs are required. Such networks can be designed by a user with sufficient understanding of the problem. Secondly, the performance of an EA designed for numerical optimisation can be validated on numerical functions with known properties. This permits a greater understanding of the optimisation process performed by the algorithm and under what conditions it may fail.

4.2 The selection of an evolutionary algorithm

ES were selected from the family of EAs for the evolution of connection weights based on the following properties:

Correlated mutations Correlated mutations permit the capture of dependencies between individual connection weights in a NN.

Self-adaptation of strategy parameters The self-adaptation of strategy parameters decreases the number of parameters that must be tuned by the user on a per problem basis. Also, it permits the algorithm to adapt to a fitness function that changes during the evolutionary process. This is an advantage when dealing with the bootstrap problem.

Real-valued representation ES are typically used for continuous parameter optimisation and are well suited to the evolution of connection weights, which are specified by a set of real-valued parameters.

ES with correlated mutations were used in early experiments, but problems were soon identified with the canonical version of the correlated mutation operator. An ES with correlated mutations must evolve $\mathcal{O}(n^2)$ rotation strategy parameters, where n is the dimensionality of the problem. These rotation strategy parameters are used to calculate a rotation matrix that rotates the mutation distribution. As the evolution of connection weights is a high dimensional problem, the number of rotation strategy parameters easily exceeds the number of connection weights being evolved. Also, the computational cost of so many successive rotations is very high and the time required to execute the mutation operator can represent a significant percentage of the total execution time. For these reasons the use of ES with correlated mutations is impractical for the evolution of NN connection weights.

Note that ES with uncorrelated mutations are still applicable to the evolution of connection weights. However, the lack of correlated mutations is viewed as a major disadvantage.

The CMA-ES was selected to replace ES with correlated mutations and was used for all experiments presented in this thesis. It does not evolve rotation strategy parameters, but rather implements a principal component analysis of the previously selected mutation steps to determine the new mutation distribution. Thus, it does not suffer from the problems associated with evolving a large number of rotation strategy parameters. Furthermore, the CMA-ES has shown very competitive performance on a suite of numerical test problems.

The use of the CMA-ES was inspired in part by the results of Igel (2003) on the inverted pendulum problem. At the time that the work in this thesis was initiated CMA-ES was the strongest performing algorithm on this benchmark problem, although it has very recently been outperformed by CoSyNE (Gomez et al., 2006, 2008). Nevertheless, CMA-ES remains a very competitive algorithm for continuous optimisation problems.

4.3 No free lunch theorem

The No Free Lunch (NFL; Wolpert and Macready, 1997) theorem for search and optimisation roughly states that "... any two algorithms are equivalent when their performance is averaged across all possible problems" (Wolpert and Macready, 2005, p. 721) or "... for any algorithm, any elevated performance over one class of problems is offset by performance over another class" (Wolpert and Macready, 1997, p. 67). This theorem remains an active area of research.

The NFL theorem raises a few questions with respect to NE. Does the NFL theorem apply to the evolution of NN connection weights? Or can an EA be found that will outperform all other EAs at finding connection weights for all problems? That is, to what degree is the search in the connection weight space influenced by the state space of the RL problem? The answers to these questions are beyond the scope of this thesis. However, questions such as these should be considered when evolving NNs using any algorithm.

So, it cannot be asserted that the CMA-ES will be the EA with the best performance for all problems when evolving NN connection weights. However, many of the properties it possesses are desirable for any EA used to evolve NNs. For example, self-adaptation reduces the number of algorithm parameters that must be set while allowing the fitness function to change during evolution and correlated mutations capture relationships between connection weights throughout the NN.

4.4 Neural network design

Small NNs with simple architectures were used in this thesis. They were fully connected with an input layer, output layer and a single hidden layer of less than ten neurons. Generally, the goal when selecting the size of the hidden layer was to provide a sufficient number of neurons such that the NN could learn the task without making the search space so large that the EA would have trouble finding a solution.

The hyperbolic tangent was used as the activation function for all neurons. Inputs to the NN were linearly scaled from the input range to $[-1, 1]$, although inputs were not constrained to these bounds. All outputs were linearly scaled and constrained from $[-1, 1]$ to the output range.

For problems involving the control of a vehicle, the inputs to the NN are often orientation angles or angular errors. As a NN will have difficult determining that angles in the vicinity of $\pm\frac{\pi}{2}$ are co-located in the state space, all angular values were input as unit vectors by taking the cosine and sine of their value. Thus, a single angular value requires two input nodes.

4.5 Evaluation noise

A major challenge when evaluating individuals during the evolution of a neurocontroller is sampling the state space such that:

- the problem can be solved;
- the solutions generalise well; and
- the evolution can be executed within a reasonable time frame with the available computational resources.

These issues are caused by the continuous and multidimensional state spaces typical of control problems where the difficulty of the problem varies across the space (in terms of the presence of nonlinearities and discontinuities).

The *evaluation set* is defined as the set of states used to evaluate a solution. As the population consists of multiple individuals that must be evaluated at each generation, many tens or hundreds of thousands of evaluations will be conducted during a typical run of an EA. Thus,

the size of the evaluation set will be limited by the computational resources available and it is normal for the evaluation set to undersample the state space. The states in the evaluation set can be either selected before the evolution and held constant throughout, or randomly sampled from the state space.

Selecting a few random states for the evaluation set of a solution each generation has the effect over successive generations of covering the entire search space as if the evaluation set of each solution had been much larger. The randomness of the evaluation sets will favour the evolution of networks that can generalise across the entire state space and disadvantage the evolution of overfitted networks. However, the disadvantage of this approach is that it will introduce *evaluation noise* into the fitness function.

Evaluation noise is caused by incomplete sampling of large state spaces where the difficulty of the problem varies across the space. For such problems, an individual evaluated at two random points within the state space may receive two different fitness values. The rank of an individual in the population is, therefore, partially dependent on the set of points within the state space at which all individuals were evaluated. For example, an inferior individual evaluated at an easy point in the state space may be ranked higher than a superior individual evaluated at a difficult point. If the individuals were evaluated at a different set of points in the state space then the ranking of the individuals in the population will change. Consequently, an EA will see the fitness function as being noisy, because multiple evaluations of an individual can return different fitness values.

The presence of evaluation noise in the fitness function can prevent an EA from finding a solution. So, it is critical that the EA used be robust to the effects of noise, but even a robust EA will fail if insufficient samples are taken from the state space. As the size of the evaluation set is increased the fitness of an individual will become more representative of its overall fitness across the state space and the amount of evaluation noise will decrease. However, completely sampling the state space is unlikely to be practical for most problems, because the computational resources required to do so within a reasonable time frame will be excessive. So, some compromise must be reached between the benefit gained from increasing the size of the evaluation set and the time required to execute the evolution.

The random evaluation set can be generated either: per individual such that each individual is evaluated at a unique set of samples, or per generation such that all individuals in the pop-

ulation are evaluated at the same set of samples. The advantage of generating the evaluation set per individual is that more of the state space is sampled per generation, which may result in fitter solutions and faster convergence times. However, the disadvantage of this approach is that inferior solutions may be evaluated at favourable states and ranked higher than superior solutions that are evaluated at unfavourable states. Initial experiments demonstrated that the best approach is to randomly generate the evaluation set once per generation and this is the approach that has been used in this thesis.

The problems created by evaluation noise can be avoided altogether by selecting a static evaluation set before the evolution and using it to evaluate each individual. This approach may be necessary if evaluation noise prevents the EA from finding a solution. The problem with evaluating each solution using the same static evaluation set is that the EA is likely to evolve solutions that are overfitted and generalise poorly for other states. This is especially a problem for EAs for two reasons. Firstly, as discussed above the required computational resources will limit the size of the evaluation set. Secondly, general solutions will not be selected above overfitted solutions because there is no fitness benefit to be gained from evolving general solutions that can solve the problem for states not included in the evaluation set.

Issues related to the use of random versus static evaluation sets are investigated in Chapter 5.

4.5.1 Practical problems

Evaluation noise also creates several practical problems for the operation of an EA. Firstly, it makes it difficult to identify the best individual found. This is because the fittest individual in the population at any generation is the one with the best fitness from a limited number of samples and not necessarily the one with the best fitness over the entire state space. Consequently, an EA cannot be terminated when an individual that satisfies some target fitness value is found and the individual with the best fitness out of all those found for each generation cannot be taken as the best one for the evolution. Secondly, the noise makes it difficult to observe the progress of an EA from a graph of fitness values at each generation. Observing the progress of an EA is useful when first attempting to solve a problem in order to determine if the fitness function, network structure, simulation model and so on are appropriate.

A Monte Carlo method (Tempo et al., 2004) can be used to overcome these problems. At every Δg generations, the mean fitness of the fittest solution is calculated from N samples at uniformly distributed points over the state space. The sampled points are randomly determined during initialisation of the EA, so as to provide a benchmark against which the performance of the individuals from different generations can be compared. These samples have no influence on the progress of the EA, but permit identification of the best individual found during the evolution and observation of the EAs progress.

The sample size is chosen according to the Chernoff bound. The Chernoff bound gives the minimum number of random samples required to find an empirical estimate $\hat{\mathbf{p}}_N(\gamma)$ of the true probability of performance $p(\gamma)$ that satisfies given reliability requirements. The probability of performance is defined as:

$$p(\gamma) = \Pr\{J(\Delta) \leq \gamma\} \tag{4.1}$$

where $J(\Delta)$ is a performance function and γ is the associated performance level. The estimate shall be within a specified accuracy $\epsilon \in (0, 1)$ from the true value with a confidence of $\delta \in (0, 1)$. Thus the random sample may fail to give a correct estimate, but will do so with a maximum probability of δ. From the law of large numbers for empirical probability, $\hat{\mathbf{p}}_N(\gamma)$ will asymptotically converge to $p(\gamma)$ as the number of samples is increased.

The Chernoff bound is given by (Tempo et al., 2004):

$$N \geq \frac{1}{2\epsilon^2} \log \frac{2}{\delta} \tag{4.2}$$

and gives $|\hat{\mathbf{p}}_N(\gamma) - p(\gamma)| < \epsilon$ with probability greater than $1 - \delta$.

When choosing the parameters $(\epsilon, \delta, \Delta g)$ a compromise must be made between reliability and time. As a general rule, confidence is cheaper than accuracy because the Chernoff bound is inversely proportional to ϵ and proportional to $\log \frac{2}{\delta}$ (Tempo et al., 2004).

For further information on Monte Carlo methods and the Chernoff bound see Tempo et al. (2004).

4.5.2 Other sources of fitness function noise

Noise that originates from within the model is classified as *model noise*. It may be caused by stochastic simulations, physically based systems or experimentally measured data. The

difference between model noise and evaluation noise is that model noise will cause an individual evaluated twice at the same point in the state space to receive two different fitness values whereas evaluation noise will not. However, the overall effect on the performance of an EA will be the same. Model noise shall not be considered in this thesis.

4.6 Evaluation set size and population factor

Two factors that influence the performance of an EA and must be set by the user on a per problem basis are:

- The size of the evaluation set. Increasing the size of the evaluation set improves the sampling of the state space, which decreases the evaluation noise.
- The population size. Increasing the size of the population improves the sampling of the search space.

Both of these factors decide the total number of solution evaluations performed by an EA each generation and thus the time or computational resources required. So, any potential performance improvements to be gained from increasing the size of the evaluation set and population must be weighed against the computational cost.

The CMA-ES calculates a recommended population size based on the number of object variables (see Section 2.1.2). However, Hansen and Kern (2004) found that increasing the population size can improve the global search performance of CMA-ES on multimodal problems. For ease of comparison with the evaluation set size, changes to the population size will be defined in terms of a *population factor* that will be multiplied by the recommended population size. Thus, changing the size of the evaluation set will have the same effect on the total number of fitness function evaluations as changing the population factor by an equal amount.

The effect of these factors is discussed in Chapter 5.

4.7 Issues related to the object variables

4.7.1 Initialisation of the object variables

The initial object variables were randomly selected from a uniform distribution with a mean of zero. The range of this distribution was found to have a significant impact on the outcome of the EA.

If the object variables, which directly map to weights and biases, are assigned large initial values then it is highly likely that the neurons in the resulting networks will be driven into saturation. If the problem cannot be solved by saturated networks then the EA must unsaturate the networks by evolving solutions with smaller object variables. This can pose a problem for EAs, such as ES, where the object variables are represented by real numbers and mutation is the primary variation operator used to generate new solutions. This is because a small decrease in the value of an object variable due to the mutation operator will not unsaturate the associated neuron and the behaviour of the network will not change. Therefore, the fitness of the solution will not be improved and there will be no selection pressure to drive the evolution of networks with smaller weights and biases. Even if the mutation step size is increased by self-adaptation of the strategy parameters, it is just as probable that the previously unsaturated neurons will become saturated as it is that the saturated neurons will become unsaturated. So, if the initial object variables are too large then the EA can be initialised in a local optimum from which it cannot escape.

From experience a uniform distribution with the range $[-1, 1]$ was found to work well for the problems studied in this work. However, in general it is suspected that the best range for the distribution will be inversely related to the size of the network layers. This is suggested by a heuristic used to initialise networks for training by backpropagation, where the range of the weights decreases as the number of inputs to the neurons increases (Haykin, 1999). Further work is required to validate this suspicion.

The initial global mutation step size σ was set to 0.2, which is equivalent to twenty percent of the object variable range. This value was found to work well with both feedforward and recurrent NNs, although no particular effort was made to find an optimal value.

4.7.2 Constraint of the object variables

Constraint of the object variables was found to be unnecessary. As discussed previously, appropriate initialisation of the object variables is the critical factor in successfully evolving NNs.

In early experiments on the Dubins Car problem (see Chapter 6) it was observed that the object variables were growing very large such that most of the evolved NNs were saturated and the EA was failing to find a solution to the problem. In response, constraints were introduced on the object variables to prevent the NNs from becoming saturated. This appeared to have a positive effect, because the EA was able to find fitter solutions than it had been before the constrains were introduced. However, the EA was still unable to find a solution that completely solved the problem. Eventually, it was realised that the EA was diverging due to low selection pressure caused by a poor fitness function. Once a better fitness function had been implemented it was found that the constrains were unnecessary and potentially even harmful for some problems. This experience illustrates the difficulty that can be experienced when troubleshooting problems with EAs due to the various components involved.

4.8 Putting it all in perspective

Claims of *biological plausibility* are sometimes invoked by authors when introducing work based on evolutionary computation. No such claims shall be made about the work in this thesis, which approaches the application of EC to the design of neurocontrollers from an engineering perspective. In engineering the worth of an algorithm for a particular class of problems must be judged based on its performance and an analysis of its strengths and weaknesses. Biology should serve as a source of inspiration, but not be used as a justification.

CHAPTER 5

Control of inverted pendulum systems

The inverted pendulum is a classical control problem that has been widely used as a benchmark for testing control algorithms in general and reinforcement learning and neuroevolution algorithms in particular. Its popularity as a test problem is due to the fact that it is an inherently unstable, underactuated and nonlinear system, which can be easily described and simulated. These characteristics are also common to underwater vehicles.

In this chapter, bulk experiments are conducted using the inverted pendulum system and the results analysed to gain insights into the approach presented in the previous chapter. These experiments are not conducted using an AUV system because the computational resources required to do so would have been excessive and a simpler system permits greater insight into the results obtained. In particular the effect of evaluation noise on the search performance of the CMA-ES is investigated. As part of these experiments, deficiencies are identified in the inverted pendulum problem benchmark and modifications to the benchmark are suggested.

5.1 Problem formulation

The classic inverted pendulum system, as shown in Figure 5.1(a), consists of a pole hinged at its bottom to a cart that is able to move freely on a track of finite length. The goal is to stabilise the pole in a vertical orientation by applying a force to the cart that will move it forwards or backwards along the track without moving beyond the track boundaries. The inverted pendulum can be characterised as an inherently unstable, underactuated and nonlinear system.

Modern control algorithms and in particular those from the field of NE are capable of solving the basic inverted pendulum problem with relative ease. For this reason several more

challenging variations of the basic problem have been proposed. The most common variation involves the addition of a second pole that is either hinged to the cart beside the first pole to create a *dual inverted pendulum* as shown in Figure 5.1(b), or hinged to the top of the first pole to create an *articulated inverted pendulum* as shown in Figure 5.1(c)[1]. The poles can only be stabilised if they are of different lengths. This is because poles of equal length are equally affected by the control force applied to the cart. For example, if both poles of a dual inverted pendulum are falling in opposite directions any force applied to the cart will drive one pole towards the desired vertical orientation while driving the other away.

Another common variation of the basic inverted pendulum problem requires that some state information be withheld from the controller, typically the linear velocity of the cart and the angular velocities of the poles. This variation is known as the *inverted pendulum without velocities* problem. It adds an additional layer of difficulty to the problem because the controller does not know the direction in which the cart is travelling nor the direction in which the poles are swinging. The controller must learn to internally estimate the missing information it requires using the current inputs and short-term memory of past inputs. This can be achieved using RNNs.

The experiments in this chapter use the dual inverted pendulum system both with and without velocities. This system is used because it is the most common variant used to benchmark NE algorithms.

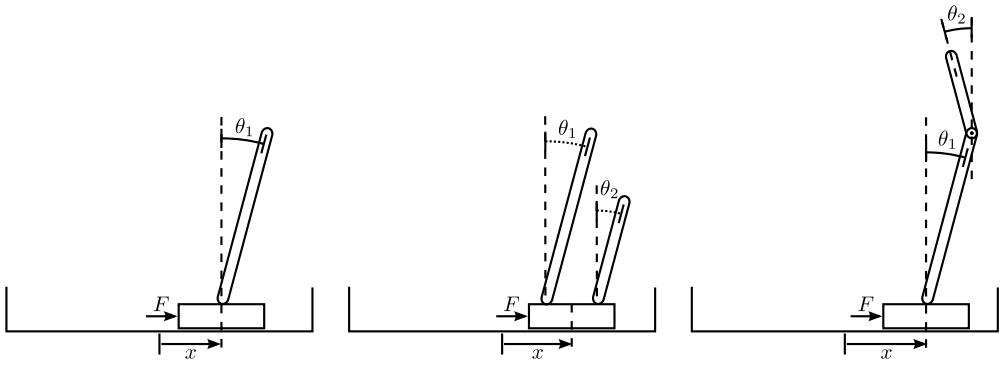
5.1.1 Dual inverted pendulum model

The dual inverted pendulum model is shown in Figure 5.1(b). The equations of motion for this model are presented in Appendix A. It is assumed that:

- the hinges between the cart and poles are frictionless,
- the interface between the cart and ground plane is frictionless, and
- the pendulum poles can be modeled as uniform slender rods.

The model parameters are given in Table 5.1.

[1]Both of these variants are also known as double inverted pendulums.



(a) Single (b) Dual (c) Articulated

Figure 5.1: Inverted pendulum configurations.

Table 5.1: Parameters for dual inverted pendulum model.

Parameter	Symbol	Value
Mass of the cart	m	1.0 kg
Mass of the long pole	m_1	0.1 kg
Mass of the short pole	m_2	0.01 kg
Length of the long pole	$L_1 = 2l_1$	1.0 m
Length of the short pole	$L_2 = 2l_2$	0.1 m
Length of the track	$L_t = 2l_t$	4.8 m
Force applied to the cart	F	$[-10, 10]$ N

Friction has commonly been included in the models used for testing of NE algorithms, but it has been deliberately excluded from this work. The frictionless assumption simplifies the model without diminishing the usefulness of the inverted pendulum problem as a benchmark. This is because friction has a damping effect that tends to reduce the amplitude of oscillations and so assist controllers to stabilise the system. Also, it should be noted that the magnitude of the friction coefficients used is typically so small that friction will have a negligible effect on the system dynamics. The exclusion of friction from the inverted pendulum benchmark problem is advocated in Geva and Sitte (1993).

The system has two equilibrium points. A stable equilibrium when the pole is hanging down and an unstable equilibrium when the poles are upright.

5.2 Criticisms of the inverted pendulum problem benchmark

Geva and Sitte (1993) identified deficiencies in the version of the single inverted pendulum problem used by earlier work to test learning algorithms for NNs. This work showed that a random search of the weight space could easily find linear controllers that balanced the pole and centred the cart on the track for a large range of initial conditions. Also, it was shown that the time to failure performance criterion did not differentiate between controllers that varied significantly in quality. Therefore, the use of an inadequate performance criterion decreased the difficulty of the single inverted pendulum problem, which was widely assumed by authors of earlier work to be a difficult non-linear problem. Geva and Sitte (1993) proposed a modified version of the problem with more stringent performance and reporting criteria that required the controllers to keep the pole upright and centre the cart on the track. This work suggested that such controllers would be harder to find by random search.

The criticisms outlined by Geva and Sitte (1993) of the single inverted pendulum problem also apply to the dual pendulum variant. Specifically, performance is measured by the time to failure and the problem can be solved using a linear controller. These issues are discussed further below.

5.2.1 Lenient performance criteria

For the inverted pendulum problem a successful controller is typically defined as one that keeps the poles balanced for some period of time. Balancing fails if the angle of the poles exceeds some failure angle or the cart hits the end of the track. This task is not strictly stabilisation control, because there is no requirement to keep the poles in an upright position nor return the cart to the centre of the track.

The time that the poles remain balanced is a poor measure of controller performance, because it does not differentiate between the various controller behaviours possible. For example, consider the following three controller behaviours:

- Indefinite bounded oscillations of the poles and cart such that the system never reaches a steady-state.
- Slowly decaying oscillations of the poles and cart until the system reaches a steady-state.

- Minimal oscillations of the poles and cart with the system reaching a steady-state in a short period of time.

All three of these controllers satisfy the time balanced performance criterion, although the nature of each controller varies from the others. The first controller is marginally stable, while the second and third controllers are asymptotically stable. From a practical control design perspective a marginally stable controller is unacceptable. To illustrate, consider the discomfort of passengers on an aircraft continuously pitching nose up and down during flight. Furthermore, the third controller is more optimal than the second controller because it reaches a steady-state in less time and with fewer oscillations. So, even though all three controllers are equal according to the time balanced performance criterion, the quality of the controllers varies greatly.

5.2.2 Linearity of the inverted pendulum system

The inverted pendulum problem is typically described as being a difficult nonlinear problem. However, for the region of the state space about the unstable equilibrium the dynamical response of the system is approximately linear. For the dual inverted pendulum system, this can be shown by linearising the equations of motion given in Appendix A about $\boldsymbol{x} = 0$ using the small angle approximation. For the region of the state space about $\boldsymbol{x} = 0$ where the small angle approximation holds[2] the dynamic response of the system is approximately linear. This analysis also applies to the single and articulated inverted pendulum systems. Thus, the inverted pendulum system can be stabilised by a linear feedback controller such as the Linear-Quadratic Regulator (LQR) for the linear region of the state space.

In the version of the inverted pendulum problem used to benchmark NE techniques it is normally assumed that the poles are balanced if the pole angles remain within the range $[-36, 36]$ degrees. As this region of the state space is approximately linear, the evolved neurocontrollers need only learn a linear control law.

Widrow and Smith (1964); Widrow (1987) demonstrated that the single inverted pendulum problem can be solved by a bang-bang controller using the linear control law:

$$F = F_{\max} \operatorname{sgn}(k_1 x + k_2 \dot{x} + k_3 \theta + k_4 \dot{\theta}) \tag{5.1}$$

[2] At $20°$ the error of $\sin\theta \approx \theta$ is less than three percent.

where $F_{\max}$ is the maximum force and k_1, k_2, k_3 and k_4 are coefficients that depend on the system parameters. As this equation describes a four input linear thresholding neuron, the single inverted pendulum problem can be solved using a single neuron (Wieland, 1991). Geva and Sitte (1993) showed that this control law can successfully balance the poles beyond the region of the state space where the linear approximation holds.

Given the linearity of the dual inverted pendulum problem, it seems reasonable to assume that it to can be solved by a similar control law with two additional terms for the second pole. This was demonstrated by Kassahun (2006) using EANT for the dual inverted pendulum with velocities problem where the best evolved neurocontroller consisted of a single neuron.

So, the inverted pendulum problem used to benchmark NE techniques is not the difficult nonlinear problem that it is typically assumed to be. This does not necessarily mean that it is an easy problem for some learning algorithms to solve. However, it does mean that stochastic search algorithms, such as EAs, that search directly in the weight space will perform very well on this problem. This is because for the smallest possible NN consisting of a single neuron only six connection weight values need to be found and if the time balanced performance criterion is used many such solutions will exist in the weight space.

5.2.3 Suggested improvements

Based on the suggestions of Geva and Sitte (1993) the following changes were made to the inverted pendulum problem benchmark used in this thesis:

- Increase the range of the pole angles for which the poles are considered balanced to $[-90, 90]$. This will permit a greater degree of nonlinearity in the dynamical response of the inverted pendulum system.

- Increase the range of the pole angles over which controllers are tested. However, the actual range over which the system can be stabilised will be set by the maximum control force that can be applied to the cart.

- Measure performance by the time required by the controlled system to reach a steady-state and the control force applied to the cart. This will permit differentiation between controllers based on quality.

5.3 Overview of the approach used by previous authors

This section shall describe the approach proposed by Gruau et al. (1996) and used by several subsequent authors (Gomez and Miikkulainen, 1999; Stanley and Miikkulainen, 2002; Igel, 2003; Kassahun, 2006; Dürr et al., 2006; Gomez et al., 2008) to the evolution of neurocontrollers for the dual inverted pendulum problem.

The approach described here has been used to benchmark most of the NE algorithms described in Section 2.3. Based on the most recently published results (Gomez et al., 2008) the NE algorithms ranked in order of performance on the inverted pendulum problem from best to worst are: CoSyNE, CMA-ES, EANT, AGE, NEATand ESP. Other, earlier work has tested GAs (Wieland, 1991) and EP (Saravanan and Fogel, 1995) on this problem. As Igel (2003) has already compared the performance of the CMA-ES to other NE techniques on this problem, such a comparison will not be repeated here. For a comparison between NE and reinforcement learning methods see Gomez (2003); Gomez et al. (2008).

5.3.1 Method

The evolutionary process was conducted using an evaluation set consisting of a single initial state: $\theta_1 = 1^\circ$ for the with velocities problem and $\theta_1 = 4.5^\circ$ for the without velocities problem. All other states are equal to zero.

A separate test was conducted to asses the ability of the fittest solutions to generalise for other initial states. At each generation a *generalisation test* was performed, whereby the fittest solution was tested at 625 points in the phase space given by $(x, \dot{x}, \theta_1, \dot{\theta}_1)$ with all other states equal to zero. The points are generated as follows (Kassahun, 2006):

$$\begin{aligned} x &= 4.32k_i - 2.16 \\ \dot{x} &= 2.70k_j - 1.35 \\ \theta_1 &= 7.2^\circ k_m - 3.6^\circ \\ \dot{\theta}_1 &= 17.2^\circ k_n - 8.6^\circ \end{aligned} \tag{5.2}$$

where $(i, j, m, n) \in \{0, 1, 2, 3, 4\}$ and $(k_0 = 0.05, k_1 = 0.25, k_2 = 0.5, k_3 = 0.75, k_4 = 0.95)$. The *generalisation performance* of a solution was the number of points in the phase space for which it could balance the poles for $1,000$ time steps (20 s). The evolutionary process

was terminated when a solution was found that balanced the poles for the evaluation set for $100,000$ time steps ($2,000$ s) and had a generalisation performance of at least 200.

It is worth noting that the range of pole angles used in the generalisation test, $\theta_1 \in [-3.24, 3.24]$ degrees and $\theta_2 = 0$, are well within the boundaries of the linear dynamical response of the inverted pendulum system. Also, the more difficult initial states where the poles are deflected in opposite directions are not included.

5.3.2 Fitness functions

Two different fitness functions have been used for the inverted pendulum with velocities and without velocities problems.

For the with velocities problem the fitness function was designed to balance the poles for the duration of the simulation. It is given by:

$$f = \frac{t_{\text{finish}}}{t_{\max}} \tag{5.3}$$

where t_{finish} is the time at which the angle of any pole exceeds the failure angle θ_{fail} and $t_{\max}$ is the maximum time that the simulation is permitted to run. Thus, $f = 1$ for solutions that learn to balance the poles.

For the inverted pendulum problem without velocities problem Gruau et al. (1996) observed that this fitness function evolves neurocontrollers that balance the pole by swinging it back and forth. Although the neurocontrollers are able to keep the pole from falling over, they do not learn to bring the pole to an upright position. This is because the fitness function does not discriminate between solutions based on how they keep the pole balanced. Once a solution has been found that keeps the pole balanced for the duration of the simulation there is no selection pressure to find a better one.

Gruau et al. (1996) proposed a fitness function to penalise the swinging of the poles with the intention of forcing the neurocontroller to learn how to estimate the velocities of the cart and poles when they are not provided as inputs. It comprises of two components: the first rewards successful balancing of the pole, while the second penalises oscillation of the cart and longest pole. Gruau's fitness function is given by:

$$f = 0.1f_1 + 0.9f_2 \tag{5.4}$$

where the two fitness components are:

$$f_1 = \frac{N_t}{1000} \tag{5.5}$$

$$f_2 = \begin{cases} 0 & N_t < 100, \\ \min\left[1, \dfrac{0.75}{\sum_{i=N_t-100}^{N_t}(|x^i| + |\dot{x}^i| + |\theta_1^i| + |\dot{\theta}_1^i|)}\right] & N_t \geq 100 \end{cases} \tag{5.6}$$

and N_t is the number of steps that the pole remained balanced.

Subsequent authors (Gomez and Miikkulainen, 1999; Stanley and Miikkulainen, 2002; Igel, 2003; Kassahun, 2006; Dürr et al., 2006; Gomez et al., 2008) have presented Gruau's fitness function without limiting the maximum value of f_2 to one. It is uncertain if this omission is deliberate or an error caused by the fact the original paper describes the fitness function, but does not present the actual equation. If the maximum value of f_2 is not limited to one then the contribution of f_1 becomes insignificant, because as the denominator of f_2 approaches zero the value of f_1 approaches infinity. However, if the limit on f_2 is included it prevents the evolution of solutions with performance better than some arbitrary level. That is, once the values of $(\dot{x}, \dot{\theta}_1, \dot{\theta}_1)$ are such that for $f_2 > 1$ no further improvement of the solution is possible. As all subsequent authors have excluded the limit on f_2, the same approach shall be followed in this thesis.

5.3.3 Analysis of the evolutionary process

Notwithstanding the success that this approach has had in evolving neurocontrollers that can balance the poles, it does so by exploiting the deficiencies in the inverted pendulum problem benchmark discussed in Section 5.2.

Consider the behaviour of an EA attempting to find a solution that will pass the generalisation test. Once the EA has found a solution that can balance the poles for the duration of the simulation from the given initial state there will be no selection pressure to find general solutions. The only selection pressure driving the search will be to maintain the ability to balance the poles for the evaluation set and to minimise oscillations of the cart and longest pole. Therefore, the EA will be limited to randomly searching the search space in the vicinity of the aforementioned solution until by chance it finds a solution that passes the generalisation test.

From the above discussion, two phases can be identified in the progress of the EA. In the first phase the EA is operating normally as a stochastic optimiser searching for a solution that can balance the poles for the evaluation set. Whereas, in the second phase the operation of the EA is similar to a random search as it searches for a solution that will pass the generalisation test. The fact that the EA is able to find a general solution by conducting what is effectively a random search indicates that the first phase has located the EA in a favourable region of the search space where such solutions can be found. This will be the case if the EA finds a linear controller for the given initial state.

Igel (2003) correctly observed that there is no selection pressure to evolve solutions that could pass the generalisation test and that a CMA-ES would decrease the global mutation step-size once a solution had been found that could balance the poles for the evaluation set. However, this behaviour was incorrectly identified as premature convergence and a minimum bound on the global mutation step-size was imposed to force the algorithm to randomly explore the search space. Although this modification permitted the CMA-ES to find a solution that passed the generalisation test, the fault lay with the approach rather than with the algorithm itself. The algorithm was correct in reducing the global mutation step-size, because the population was converging to a solution that could balance the poles for the evaluation set.

Other EAs, such as NEAT and EANT, will behave in a similar manner to the CMA-ES. This is because in the absence of selection pressure an EA will perform a random search of the search space. It is unreasonable to expect an EA to account for criteria that have no influence on the fitness of the solutions, because this is the EAs only source of information with which to direct the search. This behaviour is more easily observed in EAs with self-adaptation, such as the CMA-ES, because the strategy parameters provide additional insight into the algorithm's operation.

5.4 Overview of the new approach proposed in this thesis

The approach used throughout this thesis and applied to the inverted pendulum problem generates evaluation sets by randomly sampling one or more states from the state space. Although static evaluation sets have been applied successfully for this problem, this success has been due to specific properties of the problem that can be exploited by the EA. As these properties will

be absent in problems of more general interest, approaches that use static evaluation sets will not work and random evaluation sets must be used instead. It is, therefore, worthwhile studying the properties of random evaluation sets on benchmarks such as the inverted pendulum problem.

The fitness functions and generalisation test used by previous authors are not used in this thesis. This is because the generalisation test is based on the time balanced performance criterion and the neurocontrollers are tested over a very limited range of pole angles. Also, Gruau's fitness function does not account for the performance criteria suggested as improvements to the benchmark problem in Section 5.2.3.

5.4.1 Method

The evolutionary process is conducted using a randomly generated evaluation set. For each state in the evaluation set θ_1 and θ_2 were selected randomly with a uniform distribution in the range $[-30, 30]$. All remaining states were set to zero. Although the remaining states could have also been randomised, the use of the pole angles alone facilitates a more straightforward interpretation of the results. It also presents a problem that is sufficiently challenging, as long as the bounds of the pole angles are set large enough. This is because the pole angles are the primary source of nonlinearity in the inverted pendulum system.

The best individual found during the evolutionary process was determined using the Monte Carlo method described in Section 4.5.1. Every ten generations the individual with the best fitness value was evaluated at 185 $(\epsilon = 0.1, \delta = 0.05)$ uniformly distributed randomly generated points in the phase space $(\theta_1, \theta_2) \in [-45, 45]$ degrees. These points were generated once during the initialisation of the EA. This test is independent of the evolutionary process.

5.4.2 Fitness function

A new fitness function is proposed for the inverted pendulum problem that is designed to evolve neurocontrollers that satisfy the performance criteria discussed in Section 5.2.3. It is constructed in the form of an optimisation problem where the objective is to minimise the sum of the root mean square of the normalised cart position, pole angles and control force at each time step. Normalisation of the components is important because it alleviates scaling issues

that could cause one component to dominate the others. Furthermore, the root mean square is used instead of the mean because it penalises oscillations of the states and control force.

The greatest difficulty with this approach is how to deal with failed balancing attempts. As there is no time-based reward and the evaluation of a solution is terminated when balancing fails, the fittest solution will be that which does not swing the poles and fails quickly. To overcome this problem the maximum value of each component (equal to one because each component is normalised by its bounds) is included in the root mean square calculation for the remaining time steps after balancing fails. This ensures that successful balancing attempts will always receive higher fitness than failed balancing attempts.

So, the final fitness function is:

$$f = \sqrt{\frac{1}{N_t}\left(\sum_{i=0}^{i=N_s}\left(\frac{x_i}{l_t}\right)^2 + N_r\right)} + \sqrt{\frac{1}{N_t}\left(\sum_{i=0}^{i=N_s}\left(\frac{\theta_{1i}}{\theta_{\text{fail}}}\right)^2 + N_r\right)} + \sqrt{\frac{1}{N_t}\left(\sum_{i=0}^{i=N_s}\left(\frac{\theta_{2i}}{\theta_{\text{fail}}}\right)^2 + N_r\right)} + \sqrt{\frac{1}{N_t}\left(\sum_{i=0}^{i=N_s}\left(\frac{F_i}{F_{\max}}\right)^2 + N_r\right)} \tag{5.7}$$

where N_t is the total number of simulation steps, N_s is the number of steps that the poles remain balanced and $N_r = N_t - N_s$ is the steps remaining after balancing fails.

The advantages of this fitness function over Gruau's fitness function are: the response of the controlled system is included for all steps in the simulation; the applied control force can be optimised; and the shortest pole is included. Furthermore, the value of the fitness has a known maximum limit of 4 that occurs if balancing fails immediately.

5.4.3 Performance metrics

The performance of the evolved neurocontrollers is expressed in terms of fitness calculated using the fitness function in Equation 5.7. The fitness provides a reasonable measure of a neurocontrollers quality according the to the performance criteria proposed in Section 5.2.3. This is because the fitness function includes the state of the system, the control effort and penalises oscillations.

Figure 5.2 shows examples of the controlled response of the dual inverted pendulum system from two different initial states and the associated fitness values.

The performance of the best neurocontroller from a trial is determined by running a separate test at the conclusion of the evolutionary process. The neurocontroller is tested at 30 points in each dimension of the phase space given by $(\theta_1, \theta_2) \in [-45, 45]$ degrees. From the results for these 900 points the following metrics are calculated:

- the mean fitness for all points in the phase space,
- the failure rate defined as the fraction of points in the phase space for which balancing fails, and
- the mean fitness excluding all points in the phase space for which balancing failed.

It is not expected that the neurocontrollers will be able to stabilise the system for the entire phase space. This is because the control force is constrained.

The performance of the search process is measured by the number of fitness function evaluations required to find the best individuals. This approach is followed, rather than terminating the evolution once some arbitrary fitness level has been achieved, so as to determine the best possible performance of the neurocontrollers that can be found.

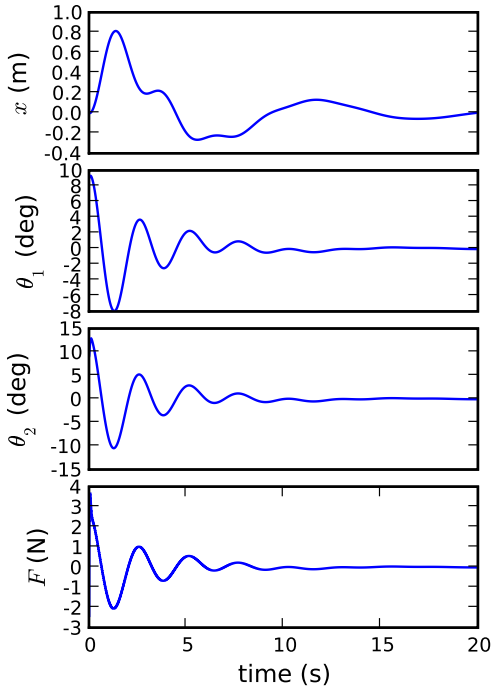
5.5 Experimental setup

The inverted pendulum equations of motion given in Appendix A were simulated using a fourth-order Runge-Kutta integrator with a time step size of $\Delta t = 0.01$ seconds. The neurocontroller was updated every 0.02 seconds. The maximum simulation time was $t_{\max} = 20$ seconds.

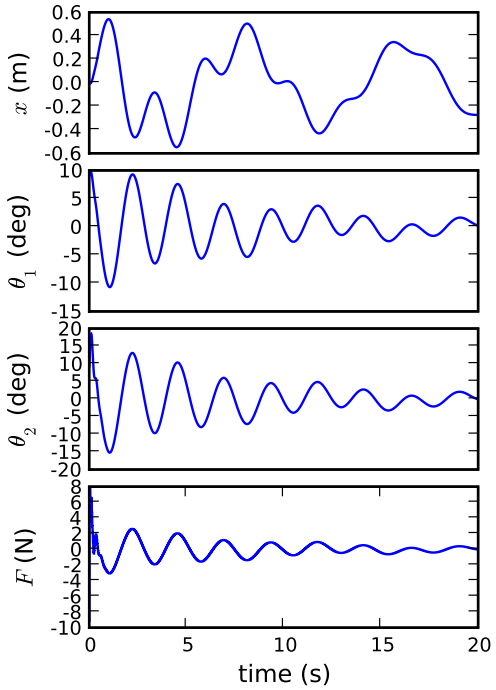
The evaluation of an individual was terminated when any of the following conditions were satisfied:

- the cart collided with the track boundaries $|x| \geq l_t$,
- the angle of any pole exceeded the failure angle $|\theta_{1 \vee 2}| \geq \theta_{\text{fail}}$, or
- the maximum simulation time had elapsed $t \geq t_{\max}$.

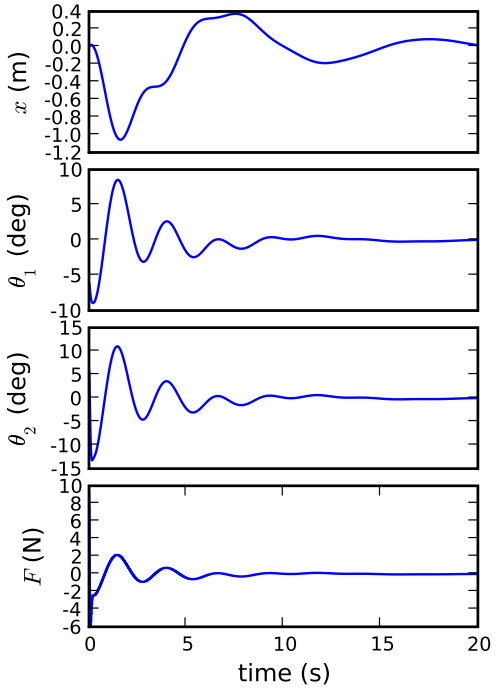
where $\theta_{\text{fail}} = 90^\circ$.



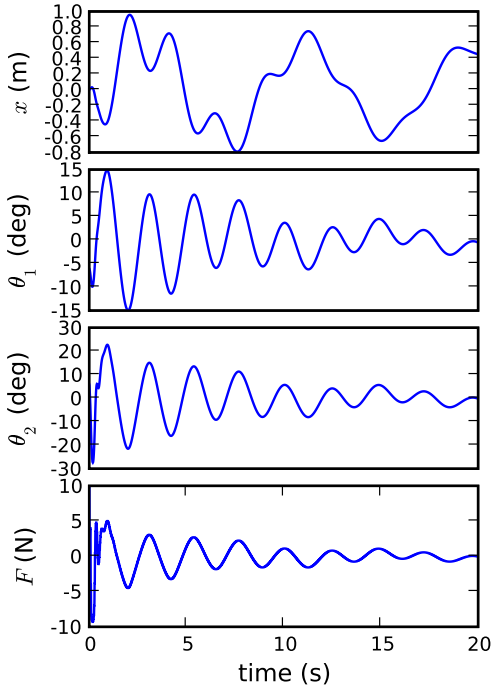
(a) $F \approx 0.21, \theta_1 = 9°, \theta_2 = 9°$



(b) $F \approx 0.33, \theta_1 = 9°, \theta_2 = 9°$



(c) $F \approx 0.28, \theta_1 = -6°, \theta_2 = 6°$



(d) $F \approx 0.52, \theta_1 = -6°, \theta_2 = 6°$

Figure 5.2: Example responses of the controlled inverted pendulum system from two different initial conditions and the associated fitness values. Examples (a) and (c) are from a neurocontroller evolved using a static evaluation set for the with velocities problem, while (b) and (c) are for the without velocities problem.

5.5.1 Design of the neurocontroller

For the dual inverted pendulum problem the neurocontroller was a fully connected multilayer feedforward NN. It had an input layer of six nodes, no hidden layer and an output layer of one neuron. The transfer function for all neurons was the hyperbolic tangent function and no neurons included biases.

The inputs to the network were:

- the position of the cart $x \in [-l_t, l_t]$ m,
- the linear velocity of the cart $\dot{x} \in [-5, 5]$ m/s,
- the angles of the two poles $\theta_1 \in [-\theta_{\text{fail}}, \theta_{\text{fail}}]$ rad and $\theta_2 \in [-\theta_{\text{fail}}, \theta_{\text{fail}}]$ rad, and
- the angular velocities of the two poles $\dot{\theta}_1 \in [-10, 10]$ rad/s and $\dot{\theta}_2 \in [-10, 10]$ rad/s

The output from the network was the force applied to the cart $F \in [-10, 10]$ N. All inputs and outputs were scaled using the given ranges according to the method in Section 4.4.

The ranges for $\dot{x}$, $\dot{\theta}_1$ and $\dot{\theta}_2$ were chosen based on experimentation. As discussed in Section 4.4, incorrectly specifying the input ranges can prevent the EA from finding a solution.

For the dual inverted pendulum without velocities problem the neurocontroller was a fully connected multilayer RNN. It had an input layer of three nodes, a hidden layer of two neurons and an output layer of one neuron. The hidden layer was fully recurrent. Otherwise all characteristics were the same as the feedforward NN described above.

Based on the results of Igel (2003) no biases were included in the NNs. Igel (2003) showed that neuron biases significantly increased the number of fitness function evaluations required to find a solution. This is due to the symmetry of the inverted pendulum system. For example, consider a linear controller consisting of a single neuron. If the poles and cart have been stabilised then the inputs to the cart are all zero and the output from the controller must also be zero. Any bias connection would prevent the NN from having a zero output in this situation. So, the increase in fitness function evaluations is caused by the need to cancel the influence of the bias connections on the output.

The design of the neurocontrollers including the number of object variables for both the with and without velocities problems are summarised in Table 5.2. As discussed in Sec-

Table 5.2: Neurocontroller properties including the number of object variables for the dual inverted pendulum with and without velocities problems.

Problem	Input nodes	Hidden neurons	Output neurons	Feedforward connections	Recurrent connections	Object variables
With velocities	6	0	1	6	0	6
Without velocities	3	2	1	8	4	12

tion 4.7.1, the object variables were randomly initialised from a uniform distribution in the range $[-1, 1]$.

5.6 Baseline performance based on a static evaluation set

A baseline against which to compare the effect of using randomly generated evaluation sets was established by experimentally determining the expected performance of neurocontrollers evolved using a static evaluation set.

The CMA-ES was used to perform 20 trials with a static evaluation set for the dual inverted pendulum problem with velocities problem. The evaluation set contained a single initial state with $\theta_1 = 4.5°$and all other states equal to zero. Each trial was permitted to run for 2000 fitness function evaluations. These trials were repeated for the inverted pendulum problem without velocities problem. However, each trial was permitted to run for 5000 fitness function evaluations. The design of the neurocontrollers used in these trials is described in Section 5.5.1.

Following the approach of Igel (2003), a lower bound was imposed on the CMA-ES global mutation step-size such that $\sigma_{min} \geq 0.05$. This constraint was necessary to prevent the CMA-ES converging with the static evaluation set and was not necessary for the random evaluation set.

The performance of the best individuals found from each trial is summarised in Table 5.3. These results show that despite being evolved at a single initial state the neurocontrollers are able to stabilise the system for a large range of pole angles with very high performance. Al-

though the failure rate may appear high, such values represent a large range of pole angles about the unstable equilibrium. This is shown in Figure 5.3, where the fitness of the best neurocontrollers for the with and without velocities problems is plotted over the phase space given by $(\theta_1, \theta_2) \in [-45, 45]$ degrees. These plots also show that the evolved neurocontrollers are much less successful at stabilising the system when the poles are deflected in opposite directions compared to when the poles are deflected in the same direction.

The neurocontrollers evolved for the without velocities problem were generally outperformed by those evolved for the with velocities problem. They tended to have a higher failure rate and also a lower performance when balancing was successful. An example of this behaviour is shown in Figure 5.2 where the response of the controlled system from different initial conditions is shown.

The performance of the search process is shown in Table 5.4. These results also include trials that failed to evolve a neurocontroller capable of balancing the poles for any points in the phase space[3]. For the with and without velocities problems there were two and five such trials respectively.

For the without velocities problem, NNs with one and three hidden neurons instead of two were also tested. Three hidden neurons were not found to provide a performance improvement, but required approximately twice the number of fitness function evaluations to find. Similarly, NNs with one hidden neuron were found to perform worse and require more fitness function evaluations. Although two hidden neurons are required to estimate the missing velocity information, NNs with one hidden neuron were tested based on the results of Kassahun (2006) where it was shown that a NN with two neurons in total was able to balance the poles albeit with poor performance. However, the evolved architecture found by Kassahun (2006) differed from that used here in that the input layer was directly connected to both the hidden neuron and the output neuron and that both neurons had self-recurrent connections. The discovery of the minimal NN architecture required to solve this problem is left as future research.

[3]They may have been able to balance the poles for the initial state in the evaluation set, because the phase space was sampled at three degree increments it was not included in the phase space.

Table 5.3: Performance of the best individuals for the dual inverted pendulum (a) with velocities and (b) without velocities using a static evaluation set calculated from 30 points in each dimension of the phase space given by $(\theta_1, \theta_2) \in [-45, 45]$ degrees.

(a) With velocities

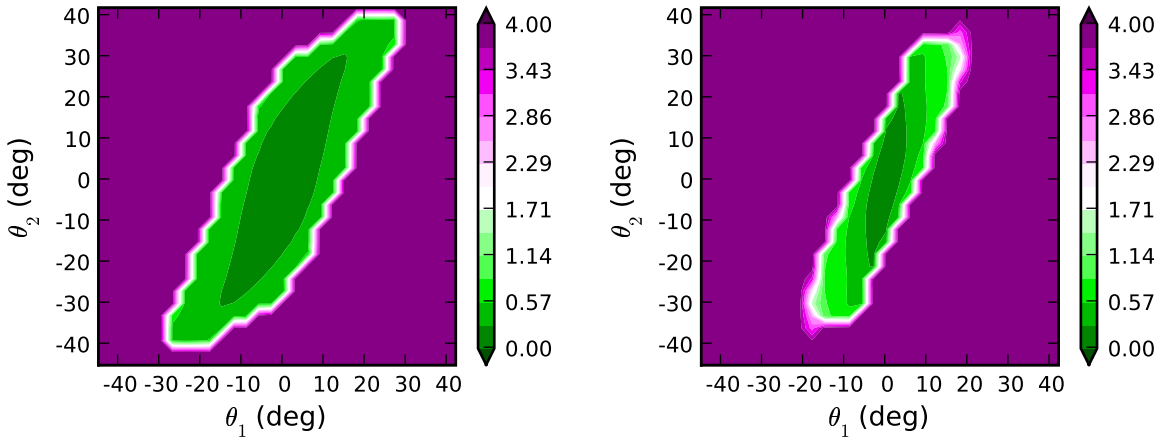
Value	Mean	Std. Dev.	Percentile		
			25th	50th	75th
Mean fitness	3.47	0.25	3.31	3.45	3.47
Failure rate	0.87	0.07	0.82	0.86	0.87
Mean fitness excluding failures	0.22	0.14	0.13	0.17	0.23

(b) Without velocities

Value	Mean	Std. Dev.	Percentile		
			25th	50th	75th
Mean fitness	3.73	0.17	3.59	3.67	3.94
Failure rate	0.93	0.05	0.89	0.92	1.0
Mean fitness excluding failures	0.39	0.2	0.29	0.4	0.45

Table 5.4: Number of fitness function evaluations required to find the best individuals for the dual inverted pendulum (a) with velocities and (b) without velocities using a static evaluation set.

	Mean	Std. Dev.	Percentile		
			25th	50th	75th
With velocities	1115.55	579.14	630.0	990.0	1642.5
Without velocities	2461.25	1550.46	1402.5	1925.0	3822.5



(a) With velocities ($F_{mean} = 3.08$).

(b) Without velocities ($F_{mean} = 3.55$).

Figure 5.3: Fitness of the best individuals found using a static evaluation set for the invented dual pendulum (a) with velocities and (b) without velocities shown over the phase space given by $(\theta_1, \theta_2) \in [-45, 45]$ degrees.

5.7 Evaluation noise and random evaluation sets

Randomly generating the initial states in the evaluation set will permit the evolution of neurocontrollers that can stabilise the system over a wider range of pole angles. However, it will also introduce evaluation noise into the fitness function that will have a negative effect on the search process. In this section, the benefits to be gained from using a random evaluation set and the negative effects of evaluation noise are experimentally determined and the results compared with the baseline established using a static evaluation set.

The experiment conducted in the previous section was repeated with a randomly generated evaluation set where θ_1 and θ_2 were selected randomly with a uniform distribution in the range $[-30, 30]$ degrees and all other states were set to zero. This range includes many initial states for which the constrained control force will prevent the neurocontroller from balancing the poles. These guaranteed failures are the primary source of noise in the fitness function for this problem. As the range of the pole angles in the evaluation set increases, the percentage of the possible states from which the poles can be balanced decreases and the fitness function becomes increasingly more noisy. The range used in this experiment was chosen in order to evolve neurocontrollers that could stabilise the system for the widest possible range of pole

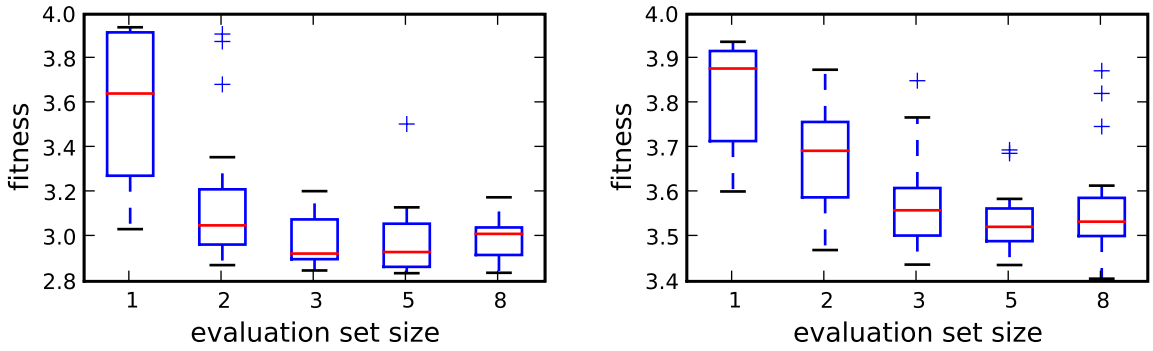
angles, but not introduce so much noise that the number of fitness function evaluations required to find a solution was excessive. The larger range of $[-45, 45]$ degrees was also tested, but was found to introduce too much noise.

The evaluation set was generated at the beginning of each generation and used for all individuals in that generation. Alternatively, the evaluation set can be generated independently for each individual in the generation. However, it was found that the associated increase in evaluation noise of a per organism approach compared to a per generation approach typically prevented the CMA-ES from finding a solution.

Twenty trials were run for both the with and without velocities problems with evaluation sizes of $\{1, 2, 3, 5, 8\}$ and a population factor of one. The neurocontroller designs are described in Section 5.5.1 and are identical to those used in the previous section. Each trial was permitted to run for $30,000$ fitness function evaluations for the with velocities problem and $100,000$ fitness function evaluations for the without velocities problem. These trials were also repeated for increasing population factors with an evaluation size of one. Compared to the trials conducted using a static evaluation set, an increase in the number of fitness function evaluations permitted for each trial was necessary so that a solution could be found.

Figure 5.4 shows the performance of the best individuals found from each trial for the with and without velocities problems. These results show that with a random evaluation set size of one the mean fitness of the evolved neurocontrollers was worse than those found with the static evaluation set. Increasing the size of the evaluation set enabled the CMA-ES to find neurocontrollers with mean fitnesses better than those found using the static evaluation set. However, as shown in Figure 5.5, this improvement in fitness came at the cost of an order of magnitude increase in the number of fitness function evaluations required to find the best solution. Furthermore, it can be observed that evaluation set sizes larger than three provided no real improvement in performance. This is may be because the best performance possible has been reached for this problem with the NNs used given the constraints placed on the control force.

Increasing the population factor was found to provide no improvement in performance when the evaluation set size was equal to one. This suggests that increasing the population factor is not an effective method of dealing with evaluation noise. Results are not shown for these experiments.



(a) With velocities

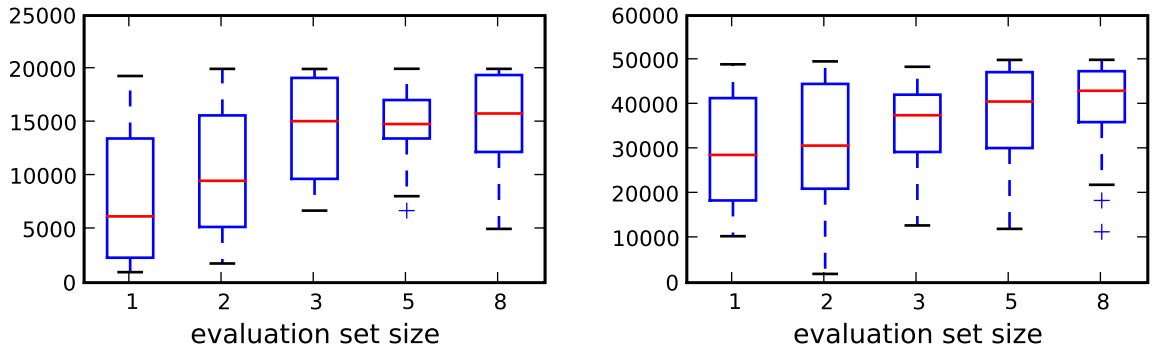
(b) Without velocities

Figure 5.4: Box plots of the mean fitness of the best individuals for the dual inverted pendulum (a) with velocities and (b) without velocities using random evaluation sets of size $\{1, 2, 3, 5, 8\}$. These values were calculated from 30 points in each dimension of the phase space given by $(\theta_1, \theta_2) \in [-45, 45]$ degrees.

Figure 5.6 shows the fitness of the best neurocontrollers for the with and without velocities problems evolved using a random evaluation set of size five and eight respectively. Compared to the best neurocontrollers found using a static evaluation set (see Figure 5.3) the region where balancing succeeds is wider, which means that the use of a random evaluation set has evolved neurocontrollers that can stabilise the system for larger deflections in opposite directions. It is also worth noting that the CMA-ES has failed to evolve neurocontrollers for the without velocities problem that match those evolved for the with velocities problem.

5.8 Discussion

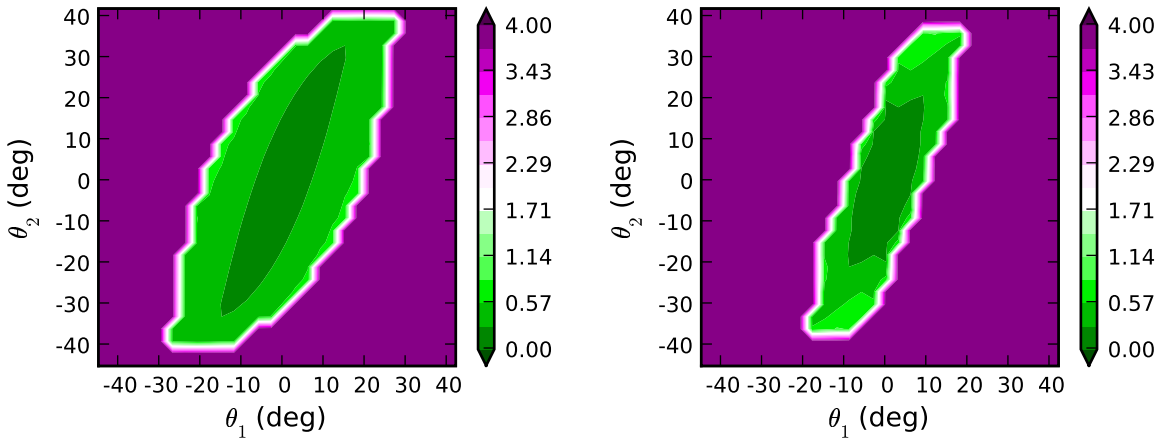
The usefulness of the dual inverted pendulum problem as a benchmark for NE algorithms is questionable. It is not the hard nonlinear problem that it is typically assumed to be and can be solved by a linear control law when all state information is provided. This linear control law is capable of stabilising the system for much of the state space and can do so with minimal control effort and with minimal oscillations of the poles and cart. As it can be implemented by a single neuron with six connection weights, NE algorithms that evolve connection weights will perform very well on this problem. Even with the improvements to the benchmark suggested in this chapter the dual inverted pendulum with velocities problem remains trivial.



(a) With velocities

(b) Without velocities

Figure 5.5: Box plots of the number of fitness function evaluations required to find the best individuals for the dual inverted pendulum (a) with velocities and (b) without velocities problems using random evaluation sets of size $\{1, 2, 3, 5, 8\}$.



(a) With velocities ($F_{mean} = 2.84$).

(b) Without velocities ($F_{mean} = 3.41$).

Figure 5.6: Fitness of the best individuals found using a random evaluation set for the invented dual pendulum (a) with velocities and (b) without velocities problems shown over the phase space given by $(\theta_1, \theta_2) \in [-45, 45]$ degrees.

The dual inverted pendulum problem becomes more difficult when velocity state information is withheld from the neurocontroller. Although the linearity of the system response does not change, RNNs are required to estimate the hidden state information. It is for this problem that the suggested benchmark improvements are most useful, because the performance of the neurocontrollers evolved without velocity information tends to be worse than those evolved with it. NE algorithms that evolve connection weights will still perform very well on this problem, so long as the size of the hidden layer is not too large.

As a consequence of the deficiencies identified in the inverted pendulum problem benchmark, it would be valuable to repeat the previously published comparisons of NE algorithms incorporating the modifications adopted in this thesis. Although such a comparison is beyond the scope of this thesis, it is expected that the modifications to the benchmark will have some effect on the outcome due to the following factors. Firstly, algorithms are ranked by the performance of the evolved neurocontrollers and the number of fitness function evaluations required to achieve this performance. Secondly, selection pressure is maintained throughout the evolutionary process and the algorithms must deal with evaluation noise. A detailed discussion of the effect of these factors on the performance of individual NE algorithms is left as an area of future research.

Evaluation noise was shown to have a pronounced effect on the evolutionary search process for this problem. At least an order of magnitude increase in the number of fitnesses function evaluations was required to evolve small neurocontrollers, represented by approximately ten object variables, when a randomly generated evaluation set was used. However, the added selection pressure also permitted the evolution of neurocontrollers with a higher performance than those evolved using a static evaluation set. Admittedly, this is an extreme example because of the inclusion of so many initial states from which the system cannot be stabilised.

The experiments in this chapter have used small neurocontrollers with no or few hidden neurons, which have been able to stabilise the inverted pendulum system for a large range of angles about the unstable equilibrium. Larger neurocontrollers capable of expressing more nonlinear behaviour were found to be unnecessary. The limited ability of the neurocontrollers to balance the poles when deflected in opposite directions was due to the constrained control force.

The performance of the evolved neurocontrollers presented in this thesis compares favourably

with the published performance of controllers designed using optimal control methods. Bogdanov (2004) compared the performance of controllers designed using a LQR, State Dependent Riccati Equation (SDRE), NN trained using back propagation through time and combinations of the NN with the LQR and the SDRE. However, a direct comparison cannot be made with these results because Bogdanov (2004) permitted an unconstrained control force that exceeded the limit imposed in this thesis by between one and two orders of magnitude when the poles were dispaced in opposite directions. Nevertheless, it can be observed that the evolved neurocontrollers perform at least as well as the LQR solutions on the dual inverted pendulum with velocities problem. This suggests that, at least for simple problems, controllers designed using NE can compete with those found using optimal control theory. A direct comparison between optimal control methods and NE on the inverted pendulum problem is left as an area of future research.

5.9 Conclusions

The inverted pendulum problem is not the hard nonlinear problem that it is typically claimed to be. Although it may be a difficult task for a human being, it can be solved by a single neuron for a large region of the state space when full state information is provided. The fact that only small NNs are required to solve the problem means that stochastic search algorithms, such as EAs, that search directly in the connection weight space will perform very well on this problem.

Two deficiencies were identified in the inverted pendulum problem benchmark: the linearity of the system about the unstable equilibrium and the large variation in controller quality permitted by the time balanced performance criterion. Modifications to the benchmark were suggested and adopted to address these deficiencies.

The inverted pendulum without velocities problem remains a useful benchmark, only if the performance of the evolved neurocontrollers is considered. It was shown that the CMA-ES is incapable of evolving RNNs for the without velocities problem that match the performance of the feedforward NNs for the with velocities problem.

Randomly generated evaluation sets were shown to improve the performance of the evolved neurocontrollers by ensuring that there was sufficient selection pressure to explore the state

space. However, the evaluation noise created by random evaluation sets was shown to increase the number of fitness function evaluations required to find a solution and to be capable of preventing a solution from being found.

Three general principles for dealing with evaluation noise are proposed. Firstly, if possible the problem should be formulated so that exceptional events, such as states for which the control problem is guaranteed to fail, are excluded from the state space. Secondly, the CMA-ES is capable of dealing with very noisy fitness functions, but an increase in the size of the evaluation set may be necessary. Thirdly, the population factor is less important than evaluation set size when dealing with very noisy fitness functions.

CHAPTER 6

Control of a Dubins car

At a basic level all autonomous vehicles must be capable of travelling from a starting location to a goal location while achieving some task or with the intention of undertaking some task at their destination. The behaviour of a vehicle driving towards a goal with the intention of arriving in a desired end state can, therefore, be considered fundamental. It can be used for waypoint following, docking with vehicles or structures, or commanded by a higher level autonomous system.

In this chapter, the neuroevolution approach is applied to the problem of a vehicle travelling to a goal in an obstacle free environment. Issues related to the formulation of neural network inputs and fitness function design are discussed that serve to illustrate general principles in evolutionary learning. Finally, the paths generated by the evolved neurocontrollers are compared with known optimal solutions.

6.1 Vehicle model

For the purposes of developing autonomous behaviours for autonomous vehicles, it is desirable to represent the motion of the vehicle with a much simplified model. This is motivated by two factors. Firstly, it is the control of the vehicle's overall behaviour that is of primary interest rather than the control of the low-level vehicular dynamics. Secondly, EAs require the vehicle simulation model to be executed many hundreds of thousands of times and for the case of a detailed model the computational resources required to do this within a reasonable time frame would be excessive. A similar approach was followed by Barlow et al. (2005) for an UAV.

The high-level kinematics of a torpedo class AUV equipped with an appropriate low-level vehicular controller can be modeled as a Dubins car. A Dubins car is capable of moving

forwards in the direction that it is facing and can turn about its vertical axis. It is unable to directly move sideways.

The use of a Dubins car model permits the low-level vehicular dynamics of the autonomous vehicle to be ignored. It is assumed that the low-level vehicular controller is capable of tracking linear and angular velocity commands while ensuring that the vehicle remains stable. Vehicle performance characteristics can be included by setting the speed range and minimum turning radius of the Dubins car model.

The kinematic equations of motion for the Dubins car model are (LaValle, 2006):

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} u\cos(\theta) \\ u\sin(\theta) \\ \omega \end{bmatrix} \tag{6.1}$$

where the vehicle state s is defined by the position (x, y) and orientation θ such that $s = (x, y, \theta)$. The control variables are the linear velocity u and angular velocity ω where $\omega \in \left[-\frac{u}{R}, \frac{u}{R}\right]$ and R is the minimum turning radius.

Typically, a torpedo class AUV will travel at a constant forward speed and is unable to turn on the spot or travel backwards. Identical constraints also apply to fixed-wing UAVs. These constraints can be modeled by setting the velocity of the Dubins car model to a constant value in the forward direction. Without loss of generality it will be assumed that $u = 1$ and $R = 1$ such that $\omega \in [-1, 1]$.

6.2 Scenario: Travel to a goal

Consider the scenario illustrated in Figure 6.1 where a vehicle must travel to a goal with the requirement that it arrive at the goal with a specified orientation ψ. The vehicle is modeled is a Dubins car travelling within a two-dimensional environment that is free of obstacles. The initial and goal states of the vehicle are defined as $s_I = (x_I, y_I, \theta_I)$ and $s_G = (x_G, y_G, \psi)$ respectively.

This scenario is a local path planning problem and is known as the Dubins car problem (Dubins, 1957).

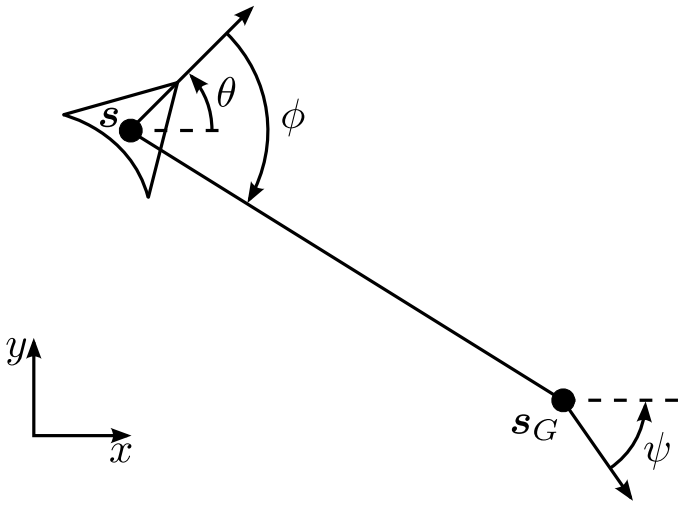


Figure 6.1: The travel to goal scenario for a vehicle modeled as a Dubins car.

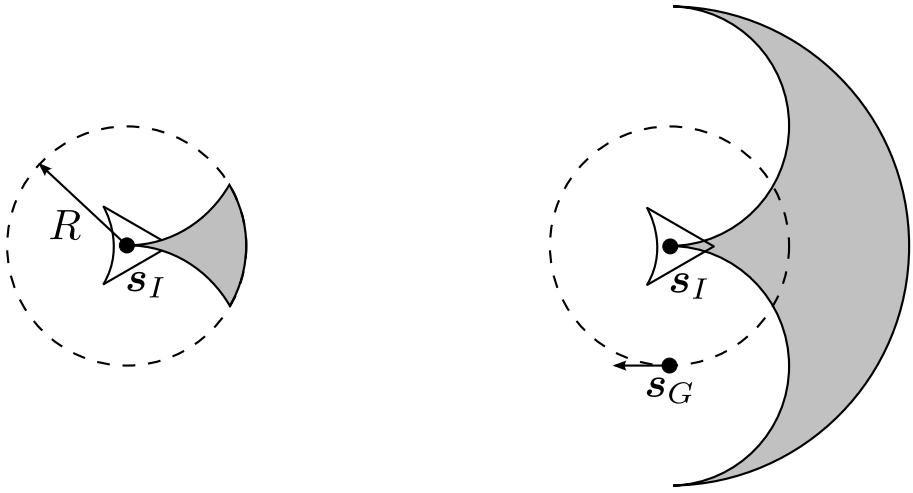
6.3 Challenges

The travel to goal scenario is challenging because the vehicle model is an underactuated system with nonholonomic constraints. Underactuated systems have fewer controls than degrees of freedom. For example, the Dubins car has three degrees of freedom (x, y, θ) and only two controls (u, ω). A nonholonomic constraint contains derivative terms that cannot be removed through integration. For example, the velocity constraints of the Dubins car are nonholonomic constraints. For a more indepth discussion of these concepts see (LaValle, 2006).

Figure 6.2a shows the reachable states for a vehicle with initial state s_I by trajectories that remain inside a circle of radius R if $u = 1$ and $R = 1$. For example, if the vehicle initial and goal states are set as shown in Figure 6.2b no trajectory exists to reach the goal within $2R$. Instead the vehicle must travel past or away from the goal in order to reach it. Given an infinitely large plane free of obstacles the vehicle will be able to reach any goal state from any initial state by performing such manoeuvres.

Dubins (1957) showed that the shortest path between any two states for a Dubins car can always be expressed as a combination of arcs of minimal turn radius and line segments. These paths are known as Dubins curves and are described in Section 6.10 where the paths generated by the evolved neurocontrollers are compared with the optimal Dubins curves.

The task is to evolve a neurocontroller that can steer a Dubins car from any initial state s_I to any final state s_G by following the shortest possible path. Given that u is a constant positive



(a) (b)

Figure 6.2: Reachable states for a vehicle with initial state s_I, where the reachable positions lie within the gray area and the reachable orientations are limited by the upper and lower curves. (a) Reachable states for trajectories that remain inside a circle of radius R. (b) No trajectory exists within $2R$ to reach the goal state s_G.

real number then the shortest possible path is equivalent to the shortest possible elapsed time.

6.4 Formulation of inputs

To solve this scenario the neurocontroller will require information about the state of the vehicle and the goal. How this information is input to the neurocontroller will at least partially depend on how it is being sensed. For example, the goal could transmit its state to the vehicle or the vehicle may determine the state of the goal using vision or sonar. For the purposes of this experiment, however, it is assumed that the vehicle has some way of determining its own state and that of the goal, that this information is free of noise and errors, and that it can be formulated in any way required before being input to the neurocontroller. Two approaches to the formulation of the inputs are compared here.

One approach is to directly provide the neurocontroller with the vehicle and goal states. The inputs to the controller will then be:

- The current vehicle state $s = (x, y, \theta)$.

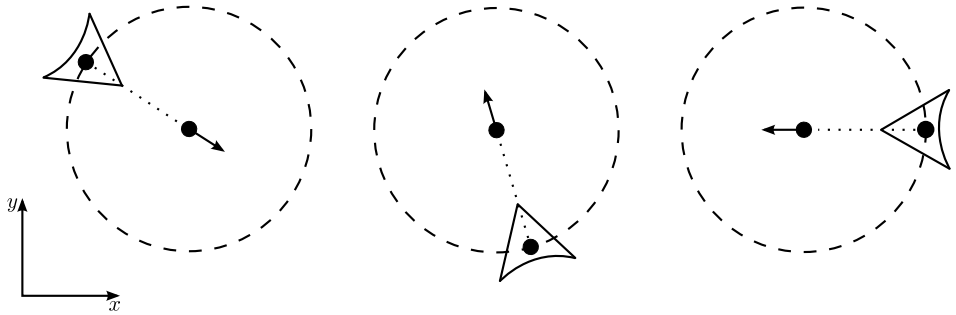


Figure 6.3: Three cases illustrating how the formulation of the inputs determines how much information the neurocontroller must learn. The vehicle is located at a point on a unit circle surrounding the goal the vehicle and goal orientations aligned.

- The goal state $s_G = (x_G, y_G, \psi)$.

This approach will be referred to as the absolute formulation.

An alternative approach is to formulate the vehicle and goal states such that the inputs to the neurocontroller are relative to the vehicle's current state. The inputs to the neurocontroller will then be:

- Euclidean distance between the vehicle and the goal d.
- Relative angle between the current orientation of the vehicle and the angle to the goal ϕ.
- Angular error between the vehicle orientation and the goal orientation $\psi - \theta$.

This approach will be referred to as the relative formulation.

Consider now the three cases shown in Figure 6.3. In each case the vehicle is located at a point on a unit circle surrounding the goal $x^2 + y^2 = 1$ with the vehicle and goal orientations aligned $\theta = \psi$. For each case the correct action by the neurocontroller is to drive the vehicle straight towards the goal.

For the absolute formulation, the inputs to the neurocontroller are different for each of the three cases shown. The neurocontroller must learn the correct action for each individual case. For the relative formulation, however, the inputs to the neurocontroller are identical for all cases. As far as the neurocontroller is concerned Figure 6.3 shows three instances of the same case, rather than three different cases that happen to have the same correct action. It needs only to learn the correct action for this one case, rather than for three. The relative formulation

of the inputs has, therefore, decreased the amount of knowledge that must be learnt by the neurocontroller as compared to the absolute formulation.

This example demonstrates the importance of applying knowledge about the problem domain when selecting the neurocontroller inputs and outputs. The correct formulation of the problem will minimise the amount of knowledge that must be learnt by the neurocontroller and make the problem easier to solve.

6.5 Method

The method used for the inverted pendulum problem in Chapter 5 was also applied here.

The evaluation set was randomly drawn each generation from a uniform distribution with $(\theta_I, \psi) \in [-\pi, \pi]$ and $d_I = 2$. Three points were included in the evaluation set because it was found that evaluation noise could prevent solutions from being found. The effect of setting d_I to a constant value on the generalisation ability of the network is investigated in Section 6.9.

The best individual found during the evolutionary process was determined using the Monte Carlo method described in Section 4.5.1. Every ten generations the individual with the best fitness value was evaluated at 185 $(\epsilon = 0.1, \delta = 0.05)$ randomly generated points. These points were uniformly distributed in the phase space given by $(\theta_I, \psi) \in [-\pi, \pi]$ radians.

The performance of the best neurocontroller from each trial was determined by running a separate test at the conclusion of the evolutionary process. The neurocontroller was tested at 50 points in each dimension of the phase space given by $(\theta, \psi) \in [-\pi, \pi]$ radians with d_I held consant. From these $2,500$ points the following metrics were calculated:

- the mean time to goal for all points in the phase space, and
- the failure rate defined as the fraction of points in the phase space for which the vehicle fails to reach the goal.

It is expected that the neurocontrollers will be able to reach the goal for the entire phase space.

6.6 Fitness function

An optimal neurocontroller will steer the vehicle from s_G to s_I by following the shortest possible path. Given that u is a constant positive real number, then the shortest possible path is equivalent to the shortest possible elapsed time. This is a minimisation problem.

A fitness function designed using only the definition of the optimal solution is:

$$f = \frac{t_{\text{finish}}}{t_{\max}} \tag{6.2}$$

where t_{finish} is the time at which the vehicle reaches the goal and $t_{\max}$ is the maximum time that the simulation is permitted to run. The total elapsed time is used in preference to the total path length because the fitness can then be normalised by a known value $t_{\max}$ and time is more convenient to measure than path length. The fittest solution will minimise the value of this function.

The weakness of the fitness function given in Equation 6.2 is that it rewards only those solutions that reach the goal. All those solutions that fail to reach the goal will receive the same fitness value of $f = 1$. As it is unlikely that the randomly generated neurocontrollers created during initialisation of the EA will reach the goal, the entire population will have the same fitness value. This is an example of the bootstrap problem and it will prevent the EA from finding a solution.

For example, a fitness function similar to Equation 6.2 has been successfully applied to the inverted pendulum problem (see Chapter 5). For that problem, a solution was rewarded for the period of time that the pendulum remained balanced before falling over. Although randomly generated neurocontrollers are only able to balance the pendulum for a short period of time, some are able to do so for longer than others. Therefore, fitness diversity existed in the population and the EA was able to gradually evolve neurocontrollers that could balance the pendulum for increasing periods of time until eventually one was found that succeeded at the task.

For this problem a fitness function is required that:

- rewards solutions that take the shortest path to the goal, and
- appropriately rewards solutions that fail to reach the goal.

Two different fitness functions are considered that satisfy these requirements.

The first fitness function is based on the principle of incremental rewards for correct behaviour. At each time step the solution gains fitness for travelling towards the goal and aligning the vehicle orientation with the goal orientation. Thus the first fitness function is:

$$f = \sum_{t=0}^{t_{\text{finish}}} \frac{|\theta - \psi|}{\pi} + \frac{d}{d_I} \tag{6.3}$$

where d_I is the initial relative distance of the vehicle from the goal.

The second fitness function adds additional terms to Equation 6.2 to assist the solution in learning to reach the goal. At the end of the evaluation, fitness is gained for minimising the remaining distance between the vehicle and the goal and the difference between the vehicle and goal orientations. Once the solution has learnt to reach the goal these additional terms will be zero and additional fitness will only be gained for optimising the path taken. Thus the second fitness function is:

$$f = \frac{t_{\text{finish}}}{t_{\max}} + \frac{\theta_{\text{finish}} - \psi}{\pi} + \frac{d_{\text{finish}}}{d_I} \tag{6.4}$$

The neurocontrollers evolved by these two fitness functions are compared in Section 6.8.

6.7 Experimental setup

The Dubins car equations of motion were simulated using a fourth-order Runge-Kutta integrator with a time step size of $\Delta t = 0.1$ seconds. The maximum simulation time was $t_{\max} = 20$ seconds.

The maximum simulation time is an important parameter for this problem. It must be much greater than the minimum time required to actually reach the goal. Otherwise, in the early generations of the evolutionary process those individuals that follow a long path to the goal will be prevented from doing so. Consequently, they will be ranked lower in the population than those individuals that drive straight towards the goal without ever reaching it. Thus, the EA is prevented from incrementally improving the paths followed by those individuals that can actually reach the goal and it will fail to find a solution.

The evaluation of an individual was terminated when the Dubins car reached the goal or the maximum simulation time elapsed. The Dubins car was considered to have reached the

Table 6.1: Number of object variables for the travel to goal scenario.

Type	Number
Feedforward weights	30
Bias weights	6
Total	36

goal if:

$$d - d_e \leq 0 \quad \text{and} \quad |\theta - \psi| \leq \psi_e \tag{6.5}$$

where d_e and ψ_e are the goal error tolerances for the distance between the vehicle and the goal and the angle between the orientation of the vehicle and goal respectively.

The inclusion of goal error tolerances was motivated by the practicalities of discrete-time simulation. Without these tolerances it would be possible for the vehicle to pass through the goal without being detected, because the vehicle state is only compared against the goal state at each time step. The magnitude of the required error tolerances is proportional to the simulation time step and the vehicle's maximum linear and angular velocities. As the goal error tolerances increase in magnitude, this problem becomes easier because the evolved neurocontroller does not need to be as finely tuned by the evolutionary process. For the work in this chapter the error tolerances were $d_e = \psi_e = 0.2$.

The neurocontroller was a fully connected multilayer feedforward NN. It had an input layer of five nodes, one hidden layer of five neurons and an output layer of one neuron. Each neuron included a bias and the transfer function for all neurons was the hyperbolic tangent function. The number of hidden neurons was selected based on results from initial experiments and an exhaustive attempt was not made to minimise the number of hidden neurons used. This issue is discussed further in Section 6.12.

The inputs to the network were $(d, \phi, \psi - \theta)$ where d is linearly scaled from $[0, 2]$ to $[-1, 1]$, but not limited to this range. The angular inputs $(\phi, \psi - \theta)$ were converted to unit vectors before being input to the network (see Section 4.4). The output from the network was ω where $\omega \in [-1, 1]$.

A summary of the total number of object variables values is given in Table 6.1.

6.8 Comparison between fitness functions

The design of the fitness function has a major influence on the success or failure of the evolutionary process. With a single scalar value it must describe the performance of every action in a control policy and the search space created by the fitness function defines the types of policies that can be found. These issues are demonstrated in this section by comparing the performance of the neurocontrollers evolved by the two fitness functions proposed in Section 6.6.

The CMA-ES was used to perform 100 trials with the fitness functions given in Equation 6.3 and Equation 6.4. Each trial was permitted to run for $25,000$ fitness function evaluations.

From this experiment it was found that the neurocontrollers evolved using the second fitness function given in Equation 6.4 consistently outperformed those evolved using the first fitness function given in Equation 6.3. This was because the first fitness function failed to reliably evolve neurocontrollers that could reach the goal for the entire state space. If the failure rate is defined as the number of points in the state space for which a neurocontroller fails to reach the goal, then only two of the one-hundred neurocontrollers generated by the second fitness function had a failure rate of zero compared with fifty-seven for the first fitness function. Note that the first fitness function can solve the problem, but does not do so reliably.

The reason for the large difference in performance between the two fitness functions can be understood in terms of the challenges described in Section 6.3. It was shown that for some (s_I, s_G) the vehicle must move past or away from the goal in order to reach it. For these points in the state space, the first fitness function punishes solutions that exhibit the correct behaviour while rewarding those that behave incorrectly. In effect, the search space created by this fitness function is preventing the EA from finding a solution due to:

- the presence of a local optimum in the fitness space that is surrounded by a large basin of attraction, and
- a mismatch between the fitness space and the solution space where solutions that are optimal in the solution space are sub-optimal in the fitness space and vice versa.

The second fitness function does not share these characteristics, because it does not specify how the vehicle should travel to the goal only that it should do so in the shortest time possible.

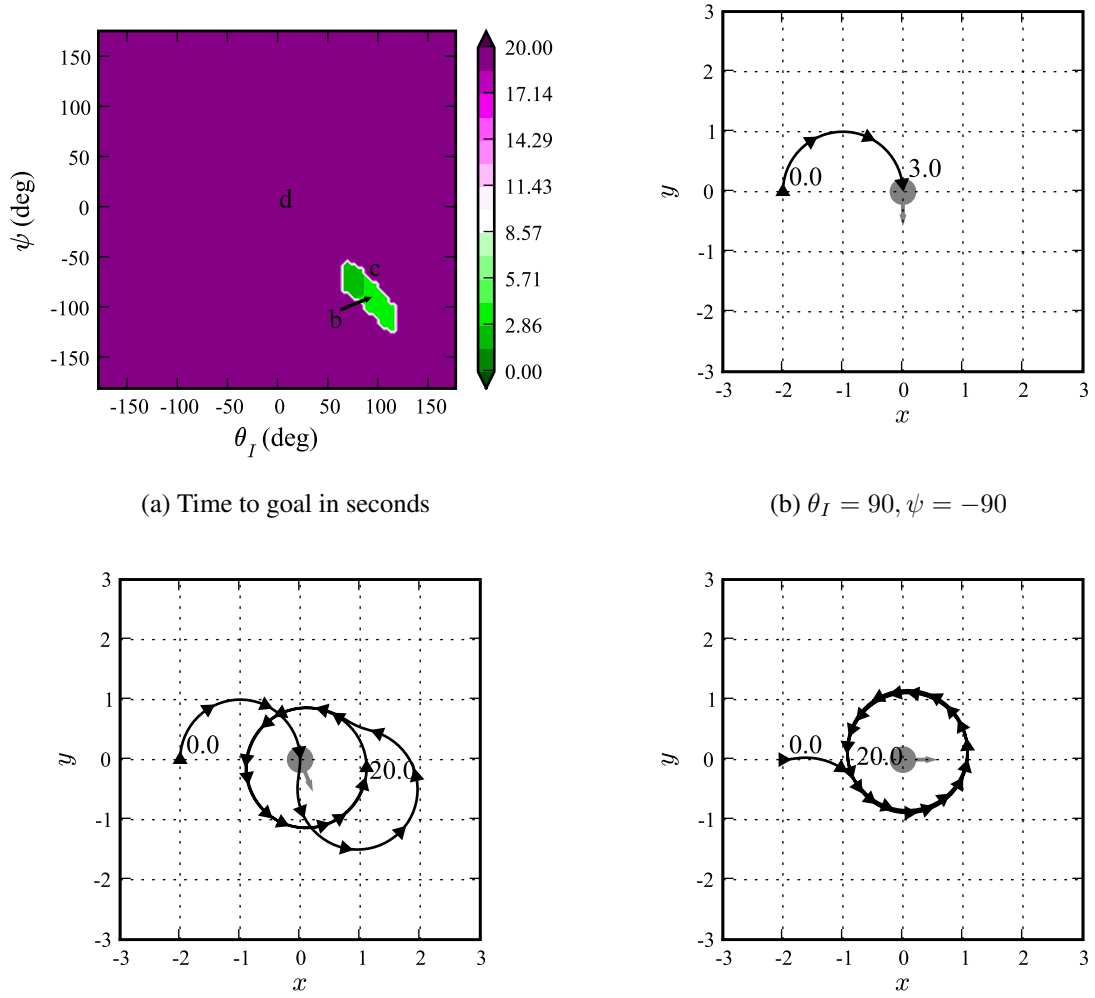
For example, Figure 6.4(a) shows the time to goal over the phase space (θ_I, ψ) for the best individual from a typical trial of the first fitness function. The neurocontroller fails to reach the goal for the majority of the phase space and is only able to reach the goal for a limited range of values centred around $(90, -90)$ degrees. This clumped structure of successful points was prevalent in the trials conducted using the first fitness function, but was not always located in the same region of the phase space. Consider the behaviour of the neurocontroller at the three points marked on Figure 6.4(a) and the corresponding trajectories shown in Figure 6.4(b-d). Point (b) shows the behaviour of the neurocontroller at $(90, -90)$ degrees where the vehicle is able to reach the goal by turning at the maximum angular velocity. However, if the point is moved to point (c) at $(90, -68.4)$ degrees the neurocontroller takes the same initial action and is unable to align the vehicle orientation with that of the goal. Failing to reach the goal state it moves through the goal, turns around and circles the goal until the maximum simulation time elapses. At point (d) located at $(0, 0)$ degrees the correct action is to drive straight towards the goal. Instead the neurocontroller takes the same action as in the two previous examples and again circles the goal. These examples show that the neurocontroller has learnt an action that works only for a small region of the state space, but not the range of actions required for the entire state space. This action represents a local optimum in the fitness space given by the first fitness function.

In the following sections, the discussion will focus on the performance of the neurocontrollers evolved using the second fitness function.

6.9 Generalisation with respect to distance from goal

In the previous section, the vehicle was initialised at a constant distance from the goal when evaluating the fitness of each individual during evolution and when testing the best neurocontrollers from each trial. The scenario, however, requires that the neurocontroller be capable of steering the vehicle to the goal from any initial state. Thus the capacity of a neurocontroller evolved with a constant value of d_I to generalise across the range of possible d_I values must be assessed.

The best neurocontrollers from the trials previously conducted using the second fitness function were tested for $d_I = 1$ and $d_I = 5$. As shown in Figure 6.5, the failure rate at both



(a) Time to goal in seconds

(b) $\theta_I = 90, \psi = -90$

(c) $\theta_I = 90, \psi = -68.4$

(d) $\theta_I = 0, \psi = 0$

Figure 6.4: Performance of the best individual from a typical trial of the first fitness function. (a) Time to goal in seconds over the phase space defined by (θ_I, ψ). (b) Trajectory at $(90, -90)$ degrees. (c) Trajectory at $(90, -68.4)$ degrees. (d) Trajectory at $(0, 0)$ degrees. The vehicle is shown at intervals of one second.

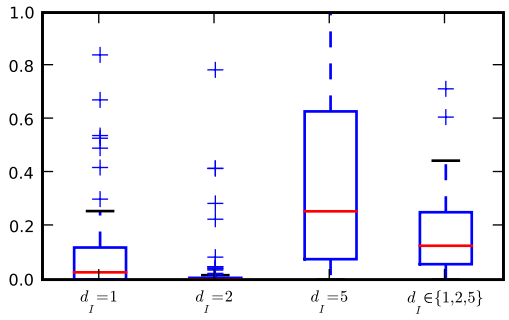


Figure 6.5: Box plot of the failure rate for the best individuals from 100 trials evolved with $d_I = 2$ over the phase space given by (θ_I, ψ) for $d_I \in \{1, 2, 5\}$.

these distances is larger than for $d_I = 2$ and the median failure rate for all distances combined was 0.2.

For this problem the training set includes all those states that the vehicle moves through when travelling from s_I to s_G, because the neurocontroller must learn the correct input-output mapping for each state along the path. As the outputs from the neurocontroller are independent of past inputs, the neurocontroller will be able to reach the goal for any s_I that lie on paths it has already learnt and it may be able to generalise for states that lie near these paths. However, states that do not lie on the fittest paths will be seen less frequently by the fittest individuals as the fitness of the population improves. The best neurocontrollers can become overtrained and lose the ability to generalise for all states when evolved using a constant value of d_I.

In order to increase the likelihood of evolving neurocontrollers that can reach the goal for all initial distances the full range of d_I values must be included in the evaluation set. This can be achieved by evaluating the individuals at randomly generated values of d_I, as has been the approach for θ_I and ψ. The cost of randomly generating d_I is an increase in the size of the state space and, hence, the amount of evaluation noise in the fitness function.

The experiment conducted in Equation 6.4 was repeated with the second fitness function. The value of d_I was selected randomly with a uniform distribution in the range $[0.1, 6]$ and the maximum simulation time was given by:

$$t_{\max} = min(20, 10d_I) \tag{6.6}$$

All other experimental parameters remained unchanged.

The best individuals from these experiments were then also tested at $d_I \in 1, 2, 5$. As shown

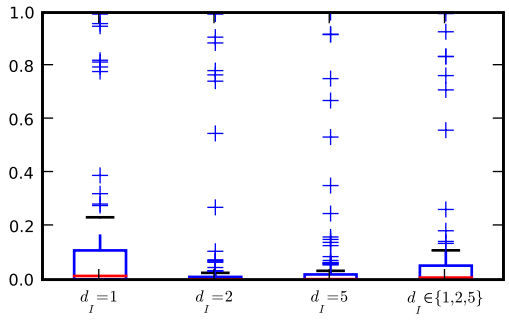


Figure 6.6: Box plot of the failure rate for the best individuals from 100 trials evolved with $d_I \in [0.1, 6]$ over the phase space given by (θ_I, ψ) for $d_I \in \{1, 2, 5\}$.

in Figure 6.6, the use of randomly generated value of d_I has substantially decreased the failure rate at $d_I = 5$ but made little difference to the failure rate at $d_I = 1$. Over all combined distances the median failure rate decreased from 0.2 to 0.01.

6.10 Comparison with optimal paths

The quality of the evolved neurocontrollers can be assessed by comparing the paths they generate with the known optimal paths for the Dubins car. This purpose of this comparison is to explore the potential of NE techniques to find optimal policies.

Dubins (1957) showed that the shortest path between any two configurations for a Dubins car can be represented as a sequence of no more than three motion *primitives*. A sequence of motion primitives is called a *word* and out of the ten possible words of three primitives only six are possibly optimal. The optimal words, known as *Dubins curves*, are LaValle (2006):

$$\{LRL, RLR, LSL, LSR, RSL, RSR\} \tag{6.7}$$

where the S primitive drives the car straight ahead at the maximum linear velocity and the L and R primitives turn the car to the left and right respectively at the maximum angular velocity. Each primitive applies a constant action over a period of time.

For convenience, the Dubins curves given in Equation 6.7 can be expressed in a more compact form by grouping paths that are qualitatively similar. The six words can be represented by two base words as follows (LaValle, 2006):

$$\{CCC, CSC\} \tag{6.8}$$

where C denotes a curve and represents either R or L.

The optimal path between any two states for a Dubins car can be determined by calculating the path length [1] of each word from the Dubins curves and choosing the word with the shortest path length. However, for real-time applications the time required for these calculations may be unavailable. Alternatively, the optimal path may be determined directly by analytically deriving the regions of the state space over which each word is optimal (Souéres and Boissonnat, 1998; Shkel and Lumelsky, 2001).

Before comparing the paths generated by the optimal Dubins curves with those generated by the evolved neurocontrollers, important differences between how they were generated must be discussed. For the Dubins curves, the control of ω is bang-bang because $\omega \in \{-1, 0, 1\}$. This differs from the evolved neurocontrollers where the control of ω is continuous because $\omega \in [-1, 1]$. Thus, the evolved neurocontrollers must learn a bang-bang control policy if they are to be optimal.

Additionally, the software used to generate the optimal Dubins curves included only the CSC paths with the CCC paths were excluded for reasons of simplifying the software implementation. However, no such constraint was placed on the evolved neurocontrollers and the evolved neurocontrollers may follow CCC paths. This is a limitation of the software used and in no way implies that the evolved neurocontrollers are superior to the analytically derived solutions.

The time to goal over the phase space $(\theta_I, \psi) \in [-\pi, \pi]$ generated using optimal Dubins curves is shown in Figure 6.7 alongside that generated using the best neurocontroller found from the trials conducted in the previous section. A comparison of these plots shows that the evolved neurocontroller has not fully learnt optimal paths. However, the structure of the time to goal plots for the evolved neurocontrollers does show some similarities with those for the optimal Dubins curves. This suggests that the evolved neurocontrollers have learnt optimal paths for some regions of the state space.

The time to goal plots for the evolved neurocontroller mostly lack the symmetry and regular structure of those for the optimal Dubins curves. This is most pronounced for the plot of $d_I = 5$, which is divided into a near optimal upper half and sub-optimal lower half by a discontinuity at $\psi \approx 0$. The regions of sub-optimal performance typically represent policies

[1] The path length of a word is equal to the sum of the lengths of all arcs and straight line segments.

where the vehicle steers towards the goal, passes it by and then turns around to reach it.

The effect of considering only the CSC paths can be observed in the plot of $d_I = 1$ for the regions of the phase space in the vicinity of $(\theta_I = 0, \psi = 0)$ and $(\theta_I = -\pi, \psi = \pi)$. In these regions, the evolved neurocontroller follows shorter paths than those generated using the optimal Dubins curves. This represents the evolved neurocontroller following the CCC paths that were excluded from the software used to generate the optimal Dubins curves. The same effect can also be observed for the plot of $d_I = 2$, but is less pronounced. Nevertheless, the fact that the time to goal plots for the evolved neurocontroller are not symmetrical shows that the evolved neurocontroller has failed to learn the optimal CCC paths for all applicable initial states.

Figure 6.8 shows a comparison between the paths generated by the best neurocontroller found in the previous section and the optimal Dubins curves for incremental values of ϕ_I where $d_I = 2$, $\theta_I = 0$ and $\psi = 0$. For six out of eight initial states (a-f) the path generated by the neurocontroller is a close approximation of the optimal path. However, for the two remaining initial states (g,h) the generated path is slightly longer than the optimal path, but the vehicle reaches the goal on the first approach.

6.11 The next step: obstacle avoidance

The next step in this work is to equip the vehicle with sensors so that it can avoid obstacles located in the environment as it travels to the goal. However, the avoidance of obstacles requires creates several problems related to the use of the sensor measurements and the associated increase in complexity.

Infrared sensors have been used by a number of authors for wall following and obstacle avoidance. The advantage of infrared sensors is that they are easily modelled and the sensor measurements can be easily input to a NN by a single input node for each infrared sensor. However, infrared sensors are of limited utility to robots outside the laboratory environment.

Sophisticated sensor systems, such as sonar and video, have received considerably less attention and they have been used the information from the sensors has typically been reduced to less than ten input values (Nelson, 2003). Two issues are inherently related to the use of sophisticated sensor systems. Firstly, large NNs with large input layers are required in order

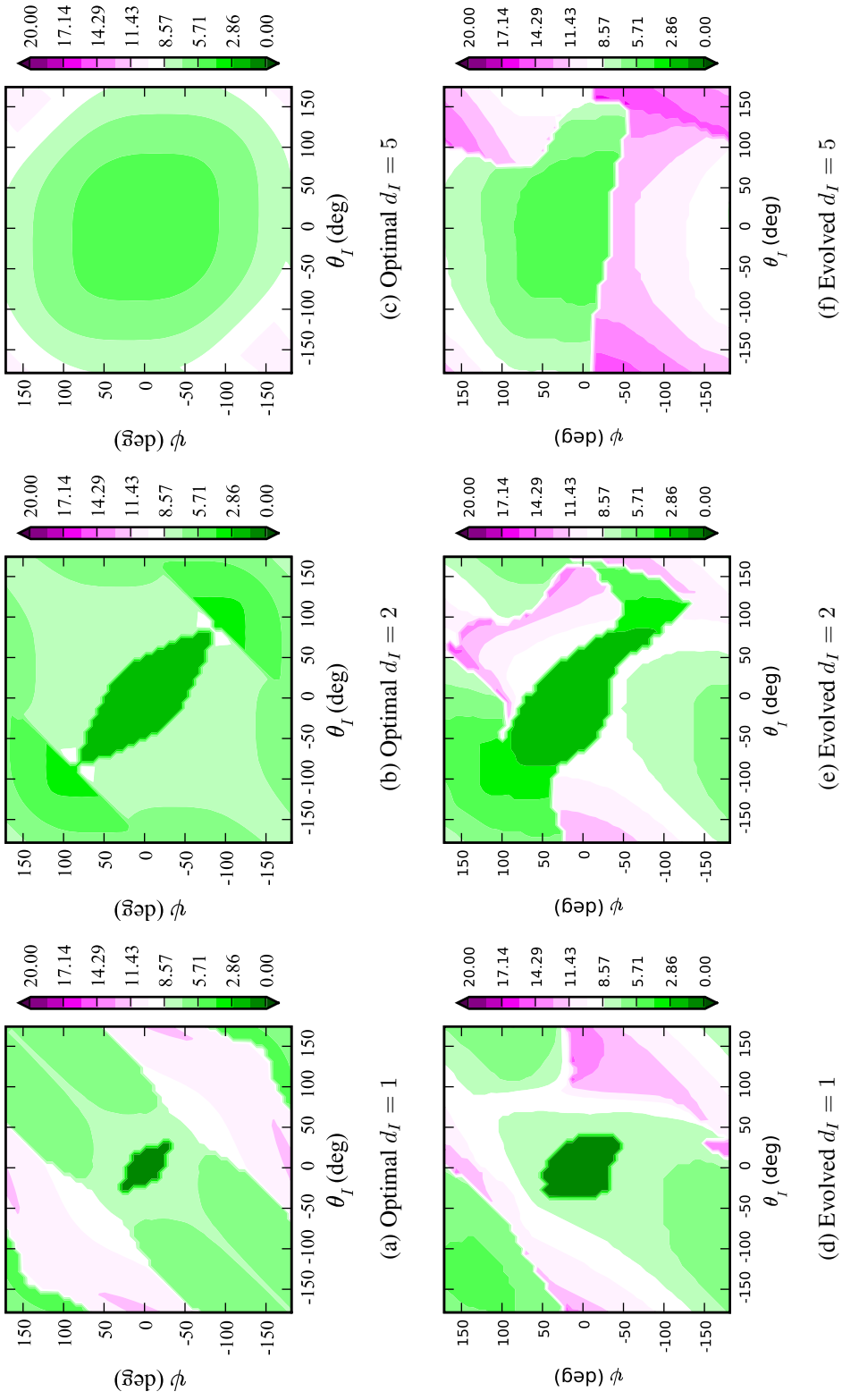


Figure 6.7: Time to goal in seconds over the phase space given by (θ_I, ψ) for $d_I \in \{1, 2, 5\}$. (a,b,c) Optimal Dubins curves excluding CCC paths (d,e,f) Best evolved neurocontroller from 100 trials evolved with $d_I \in [0.1, 6]$.

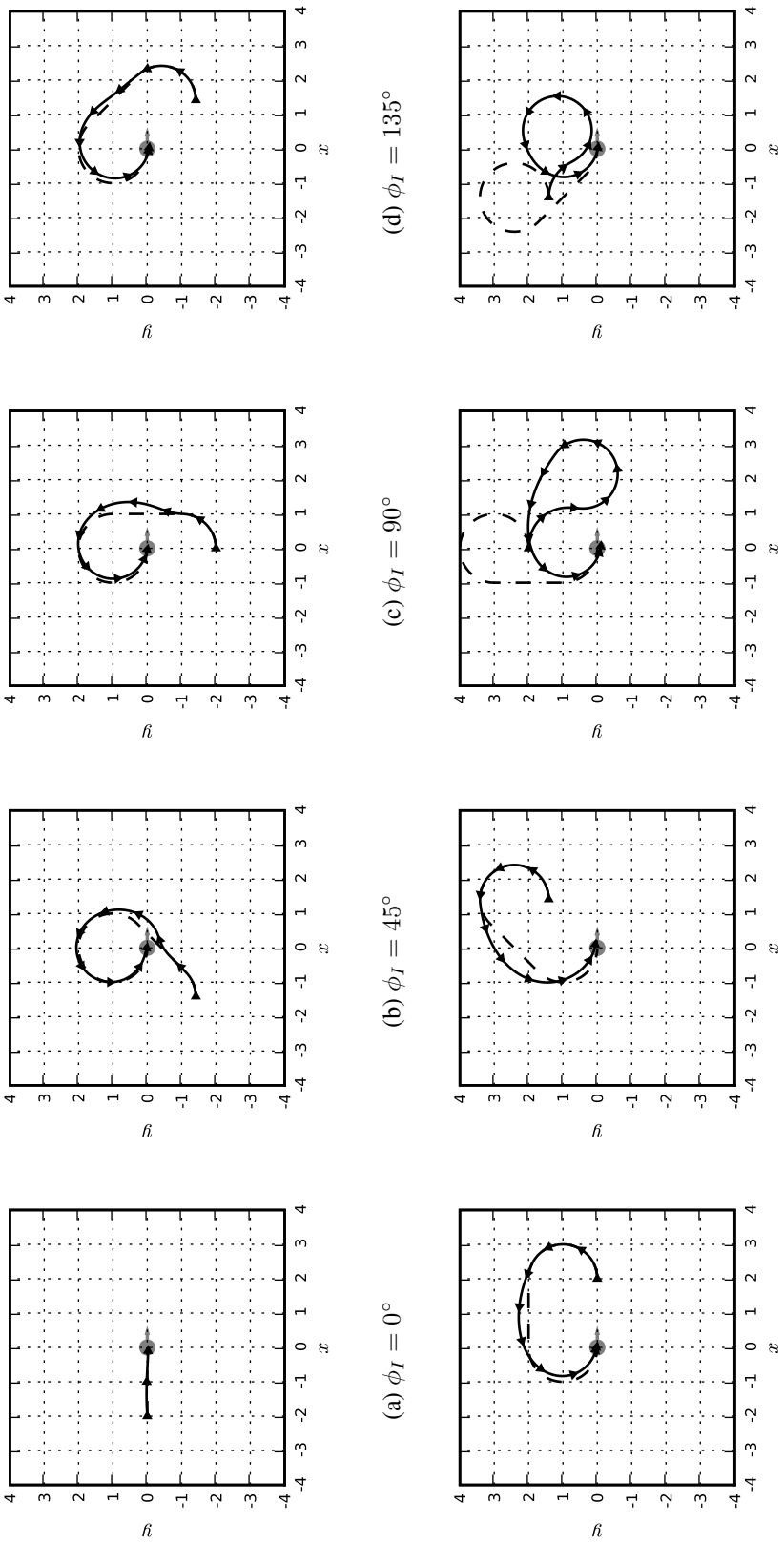


Figure 6.8: Comparison between the paths generated by the best neurocontroller found in Section 6.9 and the optimal Dubins curves, excluding CCC paths, for incremental values of ϕ where $d_I = 2$,$\theta_I = 0$ and $\psi = 0$. The optimal path is shown as a dashed line.

to receive and process the sensor information. Secondly, the NN must learn to process the raw sensor information before making use of it. The overall effect of these two issues is a considerable increase in the complexity of the problem that can prevent the evolutionary process from finding a solution.

Nelson (2003) evolved neurocontrollers for real robots in a maze that sensed the environment using colour vision. However, the video images were preprocessed to extract range data, using a method that relied upon certain fixed geometric features of the environment, that was input to the NN. Essentially, the video sensor acted as an array of thirty laser range finding sensors capable of detecting five object types. Nevertheless, Nelson (2003) did demonstrate the evolution of neurocontrollers with large input layers that were capable of navigating a maze.

If evolved neurocontrollers are to be capable of using sensor information to plan they must be able to make use of past sensor readings to build and maintain a representation of the surrounding environment. Otherwise, they will be limited to purely reactive behaviours. Nelson (2003) demonstrated the evolution of RNNs that used short term memory to escape from situations where robots may become trapped near walls. However, this is far from a demonstration of the long term memory required to plan and it is uncertain if RNNs are even capable of such behaviour.

A more realistic approach may be to have a separate process maintain a representation of the environment and then evolve neurocontrollers to operate on this information. Preliminary work was undertaken in this direction based on a potential field approach. A description of this work will be given, but results will not be presented here.

The potential field approach directs a robot as if it were "a particle moving in a gradient vector field" (Choset et al., 2005). It is a *gradient descent* method because the robot moves in the opposite direction of the gradient at its current position. Obstacles are represented by a potential field that repulses the robot, while the goal is represented by a potential field that attracts the robot. The combination of repulsive and attractive forces direct the robot from a starting position to the goal while avoiding obstacles.

In the proposed approach, the vehicle would move through a potential field with obstacles represented by repulsive spheres. The local field strength and gradient would be provided to the evolved neurocontrollers as inputs and the neurocontroller would steer the vehicle. Even though this is still a purely reactive approach, it has the advantage that the potential field can

be constructed to bias the behaviour of the neurocontroller to mimic planning.

A well known problem with potential field methods is the existence of local minima that will trap the robot (Choset et al., 2005). Local minima may be created by both concave and convex obstacles and occur at points where the repulsive fields of one or more obstacles sum to zero. The robot will become trapped in these points because no matter in which direction the robot moves it will be directed back to the local minimum point. However, it was speculated that by using RNNs such situations could be detected using short term memory and the vehicle may be able to escape.

Overall, the problem of advancing evolved neurocontrollers beyond purely reactive behaviour for path planning and obstacle avoidance remains an open research topic.

6.12 Discussion

The advantage of the approach taken in this thesis to the Dubins car problem is that the evolved neurocontrollers are closed-loop. They do not pre-generate a trajectory to the goal, but rather reactively take actions based on the current position of the vehicle and goal. This means that the neurocontrollers will be able to steer to the goal even in the presence of external disturbances, such as ocean currents. Furthermore, the evolved neurocontrollers are suitable for real-time applications where computational resources are severely limited due to cost and power constraints. This is because NNs can be efficiently implemented in software or directly implemented in hardware.

In order to improve the performance of the evolved neurocontrollers various parameter settings were tried. These parameters included: neurons without bias connections, hidden layers of ten neurons, larger evaluation set sizes and population factors, and double the number of permitted fitness function evaluations. It is possible that some untried combination of these parameters may yet solve the problem or that denser more structured evaluation sets are required to better sample the state space. Further research is required to investigate these ideas.

A similar problem to the Dubins car, which has also been used to test learning algorithms, is the truck backer-upper problem (Nguyen and Widrow, 1989). In the truck backer-upper problem the vehicle consists of a cabin and trailer connected via a hinged joint. The vehicle must be reversed into a dock such that the end of the trailer is located at the dock point and

the trailer is aligned with the dock direction. Although the truck backer-upper problem seems intuitively more difficult than the Dubins car problem, the problem can be solved using the state of the trailer only (Geva and Sitte, 1992). As the dock direction is typically fixed in a single direction, the truck backer-upper problem is a special case of the Dubins car problem.

Geva and Sitte (1992) solved the truck backer-upper problem using a linear control law implemented as a NN consisting of a single summing neuron. Such a simple solution was possible because a polar coordinate system was used to represent the problem. In contrast, earlier work by Nguyen and Widrow (1989) used a cartesian representation and required a NN consisting of twenty-five hidden neurons. These results further demonstrate the important role of domain knowledge in formulating a problem when applying any learning algorithm.

Given the results of Geva and Sitte (1992) for the truck backer-upper problem, it is possible that the Dubins car problem can be solved with fewer hidden neurons than were used to obtain the results presented in this chapter. However, as discussed above the truck backer-upper problem is a specialised case of the Dubins car problem and optimal path solutions to the Dubins car problem are nonlinear (Souéres and Boissonnat, 1998; Shkel and Lumelsky, 2001). Nevertheless, for the purpose of illustrating general principles in evolutionary learning the size of the NN used in this chapter is satisfactory. Identification of the simplest NN required to solve this problem is left as future research.

6.13 Conclusions

The evolved neurocontrollers learnt to steer to a goal in an obstacle free environment. Although the paths learnt where near optimal for some regions of the state space, the evolved neurocontrollers failed to learn fully optimal paths for the entire state space. Nevertheless, it was demonstrated that NNs can be evolved to solve this problem based on the implicit information available in the environment (in this case the minimum turning radius) and the reward from an appropriately designed fitness function.

Two general principles of evolutionary learning were demonstrated. Firstly, the amount of knowledge that must be learnt by the neurocontroller can be minimised by the correct formulation of the neurocontroller inputs and outputs based on knowledge of the problem domain. Correctly formulated inputs and outputs make the problem easier to solve by reducing the

amount of knowledge that must be learnt by the NN. Secondly, the fitness function is the problem and defines the types of policies that can be found. It must describe the performance of every action in a control policy with a single scalar value. Poorly designed fitness functions can create strong local optima in the fitness space and create a mismatch between the topography of the fitness space and the solution space.

CHAPTER 7

Framework

The preceding chapters have shown that neuroevolution is far from an automatic design process. In particular, design of a suitable fitness function, implementation of a simulation environment and tuning of the evolutionary algorithm can all require considerable effort. Nevertheless, neuroevolution remains a useful method for solving reinforcement learning problems when applied prudently.

This chapter presents a framework to guide the practical application of neuroevolution to reinforcement learning problems. Each section presents a problem that was faced while conducting the work in this thesis and, where possible, heuristic solutions are suggested.

7.1 Random evaluation sets and evaluation noise

Random evaluation sets can improve the performance of solutions to problems that feature continuous multidimensional state spaces where the difficulty of the problem varies across the state space (in terms of the presence of nonlinearities and discontinuities). Over successive generations, randomly generating the evaluation set has the effect of covering the search space as if the size of the set had been much larger. This creates selection pressure that favours the evolution of solutions that generalise well.

The benefits provided by the use of random evaluation sets comes at the cost of introducing evaluation noise into the fitness function. Evaluation noise is seen by an EA because multiple evaluations of a solution at random points within the state space return different fitness values. This makes the rank of a solution in the population partially dependent on the set of points within the state space at which all solutions are evaluated. Evaluation noise was shown to increase the number of fitness function evaluations required to find a solution. Also, the EA

will be prevented from converging to a solution if the evaluation noise is excessive.

Several factors must be considered when dealing with evaluation noise. The most critical of these factors is the selection of an EA that is robust to the presence of noise in the fitness function. As was shown by the work in this thesis, the CMA-ES is capable of dealing with very noisy fitness functions. Furthermore, the evaluation set must contain more than one member but there is a limit to the benefit that can be gained from increasing the number of members. In practice, a compromise must be reached between the benefit gained from increasing the size of the evaluation set and the time required to execute the evolution. Generally, the size of the evaluation was found to be more important than the population factor when dealing with very noisy fitness functions.

The random evaluation set should be generated once per generation and used to evaluate all solutions in the population. Although the evaluation set can be generated once per solution, it was found that doing so generally prevented the EA from finding a solution. Lastly, the problem should be formulated so that states where any solution is guaranteed to fail are excluded from the state space. This was found to be a significant problem for the inverted pendulum problem (see Chapter 5) when initial states were included in the evaluation set from which the pole could not be balanced.

The decision to use random evaluation sets must weigh the potential benefits against the known costs in the context of the problem under consideration. Generally, random evaluation sets will be necessary for RL problems because such problems are characterised by large multidimensional state spaces. For example, both the invertend pendulum problem (see Chapter 5) and the Dubins Car problem (see Chapter 6) required the use of random evaluation sets to evolve general neurocontrollers.

7.2 Problem complexity and solution proficiency

Generally, the evolutionary process will fail if the problem complexity greatly exceeds the solution proficiency. This problem is known as the bootstrap problem. It most commonly occurs when the problem is too difficult for the randomly generated solutions in the initial population. As a consequence, none of the solutions will gain fitness and in the absence of selection pressure an EA will perform a random search of the search space.

Overcoming the bootstrap problem requires that the problem difficulty be gradually increased as the solutions in the population become more proficient. This can be achieved by scaffolding the problem. Scaffolding is a temporary framework that is used to support solutions during evaluation that is gradually removed during the evolutionary process. It can be applied at three levels: by adding terms to the fitness function to reward partial successes or particular behaviours; by decomposing a complex task into easier sub-tasks; and by simplifying or modifying the environment and agent. Scaffolding a problem is a non-trivial task that requires specialised knowledge of the problem and solution domains.

For example, the fitness function used to solve the Dubins Car problem (see Chapter 6) rewarded solutions for finishing near the goal in addition to rewarding solutions for reaching the goal in the shortest possible time. Without rewards for partial success, the complexity of the problem would have exceeded the proficiency of the solutions because the randomly generated solutions in the initial population were generally unable to reach the goal. Once the solutions had learnt to reach the goal, however, the rewards for partial success became zero and additional fitness could only be gained by optimising the path taken. This is an example of scaffolding the fitness function by adding terms to reward partial success.

7.3 The role of specialised domain knowledge

In practice, the specialised domain knowledge of a human designer is required when applying NE to any RL problem. Three tasks demand a thorough understanding of the problem: designing the fitness function, scaffolding the problem and formulating the NN inputs and outputs. The specific nature of the solution to a problem may not need to be known in order to apply NE. However, this benefit is conditional on the degree of scaffolding required by the problem because scaffolding the problem requires some knowledge of the possible solutions.

For example, in Chapter 6 knowledge of the problem domain was required to correctly formulate the inputs and outputs of the evolved neurocontroller. The correct formulation of the problem minimised the amount of knowledge to be learnt by the neurocontroller and made the problem easier to solve.

7.4 Implicit versus explicit fitness functions

The choice between implicit and explicit fitness functions is a dilemma that will be faced when applying NE to RL problems. Explicit fitness functions increase the possibility that a specific desired behaviour will be evolved, but decrease the possibility that novel or unexpected solutions will be found. However, how realistic is it to expect that meaningful, unexpected solutions might be found to real problems using implicit fitness functions? Although this is one of the objectives of NE, it is typically necessary to resort to explicit fitness functions that disserve this objective if solutions are to be found in a timely manner. Generally, the work in this thesis suggests that explicit fitness functions reflect the reality of fitness function design for the practical application of NE.

The implicit versus explicit dilemma was demonstrated by the inverted pendulum problem in Chapter 5. The time balanced fitness function is an implicit fitness function that captures the overall goal of preventing the pole from falling over without constraining the possible solutions. However, this implicit fitness function evolves inferior solutions that balance the pole by rapidly swinging it back and forth. An explicit fitness function is required to find superior solutions that bring the pole to an upright position. This explicit fitness function constrains the possible solutions by including terms for the cart position, pole angles and control force.

7.5 The fitness function and search space topography

From the perspective of an EA the fitness function defines an optimisation problem. In effect, it is the fitness function and not the control task that *is* the problem. The fitness function indirectly maps connection weights to a singular scalar value via the actions in a control policy. In doing so, the fitness function defines the topography of the search space and consequently the set of all possible solutions that can be evolved. This search space will possess several local optima in which the EA can become trapped and a global optimum that corresponds to the fittest solution.

Problems arise when the designer of the fitness function is unaware that the fittest solutions defined by the fitness function do not match the solutions desired when the fitness function was

defined. Irrespective of the intentions of the designer, the EA will find the solutions that the fitness function is telling it to find. This is a common problem because the fittest solutions are often not immediately obvious from the definition of the fitness function.

For example, in Chapter 6 neurocontrollers were evolved for the Dubins Car problem using two different fitness functions. It was found that neurocontrollers evolved using one of the fitness functions consistently outperformed those evolved using the other. This was because the underperforming fitness function unintentionally punished solutions that exhibited optimal behaviour while still rewarding solutions that actually solved the problem. The EA was prevented from finding an optimal solution by the search space created by this fitness function.

A poorly designed fitness function is the most common cause of failure when attempting to evolve a solution to a problem. Visualising the fitness function can clarify the nature of the search space it defines and assist in identifying problems. Furthermore, the components of the fitness function should be normalised so that the contribution of one component to the fitness does not make the contribution of the other components insignificant. Although the contribution of each component in the fitness function can be adjusted using weight coefficients, it is typically difficult to set these weight coefficients.

7.6 Design of the simulation environment

The evolutionary process will exploit any errors or simplifications that are present in the simulation environment. This will result in solutions that *cheat* in order to gain fitness, but are of limited utility outside the simulation environment. It is critical that the behaviour of evolved solutions within the simulation environment be skeptically observed in order to detect any cheating.

Generally, simulations should be made only as complex as they need to be or, in other words, as simple as possible. This is because the potential for the evolutionary process to exploit some aspect of the simulation increases with its complexity. If necessary, additional complexity can be gradually introduced once a solution has been found using the simpler simulation. A further benefit of keeping the simulation simple is that the evolutionary process can be executed in a reasonable time frame. This is especially important given the need to observe the behaviour of the evolved solutions on a regular basis.

The risk of using simple simulations is that the evolved solutions may not transfer to the real world. Hence, it is tempting to attempt the creation of high fidelity simulations when applying NE to the design of robot control systems. However, creating such simulations is very difficult, time consuming and, ultimately, perfect simulations of the real world do not exist. Instead, the goal must be to isolate the evolved solutions from any complexity that is not being modeled. For example, an AUV was simulated using a high-level kinematic model in Chapter 6 based on the assumption that a low-level vehicular controller was available to handle the low-level vehicular dynamics.

The difference between the use of a simulation to test solutions and the use of a simulation to evolve solutions is worth highlighting. When evolving solutions, subtle errors will be detected and exploited because of the selection pressure that over many thousands of evaluations drives the creation of fitter solutions by any means possible. This selection pressure is absent when when testing solutions and so subtle errors are less likely to be detected and exploited. For example, computational stability issues were exploited by the EA during early work on the inverted pendulum problem conducted using a rigid body dynamics simulation (see Chapter 3). The upside of this property of the evolutionary process is that it can be used to detect errors in a simulation before using it to test a solution. So, while it may be desirable to test a solution using a high fidelity model it is not desirable to evolve solutions using one.

7.7 Evolution of architecture versus connection weights

The choice between NE methods that evolve NN architectures and those that evolve NN connection weights is not a straightforward one. The evolution of NN architectures the user from performing this task upfront, but if fully connected NNs will suffice for the problem then the additional complexity introduced by these by these methods cannot be justified. Conversely, the evolution of NN connection weights permits the use of methods specifically designed for numerical optimisation.

The approach advocated in this thesis is that NE is an optimisation process performed primarily on the connection weights of a NN. This approach differs from the majority of recent research in this field, which has focused on developing new algorithms for the evolution of NN architectures. Although a range of problems have been used to demonstrate the performance

of these new algorithms, there is only limited published research that directly compares their performance with that of algorithms that evolve NN connection weights. It is therefore difficult to clearly identify the performance advantages offered by the evolution of NN architecture over the evolution of NN connection weights. This problem is exacerbated by the deficiencies in the inverted pendulum problem used to benchmark all of these algorithms.

Many important questions remain unanswered regarding the advantages offered by methods that evolve NN architectures. Can they solve problems that other methods cannot? Do they find better performing solutions? Do they find solutions with fewer function evaluations? If not, what is the cost of evolving NN architectures and how does this cost scale with problem complexity? Researchers considering undertaking work in the field of NE are strongly encouraged to consider these questions.

CHAPTER 8

Conclusions

8.1 Summary of findings

The practical application of NE to RL problems was studied in this thesis. Bulk experiments were conducted on a set of problems and the results from these experiments analysed to draw out practical characteristics of NE. Based on this analysis, a framework was derived in the context of guiding practitioners and future research in this field.

Early work by the author indicated that the ER approach of evolving pure sensorimotor controllers, which had previously been applied only to simple robots operating in simple environments, is not directly suitable for application to small low-cost AUVs. However, some of the principles of ER could be applied to control problems within the reactive or behaviour-based layer of a hybrid control architecture. This early work also reinforces the importance of simulating the vehicle and environment at the highest level possible and incrementally introducing complexity to the simulation as controllers evolve with competency at less-complex tasks.

An approach to the design of NNs using EC is presented that applies an EA from the published literature designed to solve numerical optimisation problems. It is argued that NE should be viewed as an optimisation process performed primarily on the connection weights of a NN. The advantage of this approach is that it permits a deeper understanding of the optimisation process actually performed by the EA. Although the possible benefits of specialised EAs designed to evolve NN architectures are recognised, it is much more difficult to observe the optimisation process that they perform. Furthermore, the performance of a numerical EA can be measured on functions with known properties and objective comparisons made between

competing algorithms. This is not as easily achieved with specialised EAs that evolve NN architectures.

ES were selected from the family of EAs in the literature, as being the most suited to the evolution of NN connection weights. Three major advantages of ES were identified. Firstly, correlated mutations permit the capture of dependencies between connection weights. Secondly, the self-adaptation of strategy parameters decreases the number of parameters that must be tuned by the user on a per problem basis and permits the algorithm to adapt to changes in the fitness function during the evolutionary process. Thirdly, ES are designed for continuous parameter optimisation and use a real-valued representation of the object variables. However, early experiments demonstrated the weakness of ES with correlated mutations on high dimensional problems. Consequently, a variant ES known as the CMA-ES was selected and used for all subsequent experiments.

Sampling a continuous and multidimensional state space of variable difficulty was identified as a major challenge when applying NE to RL problems. The state space must be sampled such that the evolved solutions will generalise and the evolution can be executed within a reasonable time given the computational resources available. In order to satisfy these requirements the size of the evaluation set used to evaluate solutions must be kept as small as possible and the states in the evaluation set must be randomly generated each generation. Over successive generations, this approach has the effect of covering the search space as if the evaluation set had been much larger and favours the evolution of solutions that generalise well. However, the disadvantage of this approach is that it will introduce evaluation noise into the fitness function.

The inverted pendulum problem benchmark was shown not to be the hard nonlinear problem that it is typically claimed to be. Specifically, two deficiencies were identified with the benchmark: the linearity of the system about the unstable equilibrium and the large variation in controller quality permitted by the time balanced performance criterion. The strong performance of NE on this problem, as compared to other methods, can be explained by the fact that only small NNs are required and that NE is a stochastic optimisation algorithm that searches directly in the connection weight space. Modifications were made to the benchmark that addressed these deficiencies. Consequently, the without velocities variant of the problem remains a useful benchmark when the performance of the evolved neurocontrollers is considered.

Two practical characteristics of NE were explored by the bulk experiments on the inverted pendulum problem. Firstly, randomly generated evaluation sets are required to provide sufficient selection pressure such that an EA explores the state space. Secondly, the evaluation noise created in the fitness function by randomly generated evaluation sets can increase the number of fitness function evaluations required to find a solution or can prevent a solution from being found. General principles for dealing with evaluation noise were proposed.

Neurocontrollers were evolved to steer a vehicle, which was modeled as a Dubins car, to a goal in an obstacle free environment. The paths generated by the neurocontrollers were compared with known optimal solutions. Although the evolved neurocontrollers followed near optimal paths to the goal for some regions of the state space, they failed to learn fully optimal paths for the entire state space. Nevertheless, it was demonstrated that neurocontrollers can be evolved to solve a local path planning problem.

The Dubins car problem experiments demonstrated an additional two practical characteristics of NE. Firstly, knowledge of the problem domain can permit the neurocontroller inputs to be formulated in such a way as to reduce the amount of knowledge that must be learnt by the neurocontroller and, thereby, make the problem easier to solve. Secondly, the fitness function defines, often indirectly, the types of control policies that can be evolved. Consequently, a poorly designed fitness function can create a mismatch between the fitness space and the solution space such that solutions with high fitness are evolved that do not actually solve the problem.

A framework to guide the practical application of NE to RL problems is presented. This framework covers seven critical issues that were identified during the work on this thesis and can be summarised by the following advice to the practitioner:

- Use random evaluation sets to create selection pressure that favours the evolution of neurocontrollers that generalise well, but be aware of the negative effects that evaluation noise will have on the performance of the EA.

- Scaffold the problem to provide a gradual increase in problem complexity as the neurocontrollers in the population become more proficient.

- Knowledge of the problem domain is critical. Use it to design the fitness function, scaffold the problem and formulate the NN inputs and outputs.

- Be realistic about the potential of implicit fitness functions to find meaningful novel solutions to real problems. Don't hesitate to use explicit fitness functions in order to solve problems in a timely manner.
- Be aware of the search space defined by the fitness function because it defines the set of all possible control policies that can be evolved. Ensure that the fittest solutions defined by the fitness function match the desired solutions.
- Skeptically observe the behaviour of the evolved neurocontrollers within the simulation because the evolutionary process will exploit any errors or simplifications that are present. Keep simulations only as complex as they need to be and isolate the evolved neurocontrollers from any complexity that is not modeled.
- Carefully consider the choice between NE methods that evolve NN architectures and those that evolve NN connection weights. Not all problems require the additional complexity introduced by the evolution of NN architecture.

8.2 Future research

Despite the various algorithms proposed for the evolution of NNs and their many applications, evolutionary learning systems are still not fully understood. Further research is required to characterise evolutionary learning better and to identify the types of problems for which it is best suited. This thesis has contributed to this goal by critically analysing the optimisation process that occurs during evolutionary learning on two test problems.

Three studies are identified that would contribute towards an increased understanding of NE. Firstly, the relative merits of methods that evolve NN architectures and those that evolve NN connection weights must be established. Ideally, problems would be identified that either one of these approaches could not solve. Such a result would assist in determining when the additional complexity introduced by the evolution of NN architectures is justified. Secondly, a rigorous comparison is required between NE and RL in order to gain a greater understanding of their respective strengths and weaknesses. A good starting point for this comparison is the issue of trial-and-error learning that was discussed in Section 2.4. Thirdly, establish the capability of NE to find optimal solutions by comparing the performance of controllers designed

using NE and controllers designed using methods from optimal control theory. The purpose of these comparisons should not be to promote one method over the other, but rather to identify the properties of problems for which each method is best suited.

New benchmarks must be designed to perform these studies that pose a greater challenge than that of the inverted pendulum problem benchmark. Additionally, it is worthwhile revisiting previously published comparisons of NE algorithms and incorporating the modifications to the inverted pendulum problem that were adopted in this thesis.

An application area worthy of future research in ER is the evolution of neurocontrollers that are capable of long term planning. This is a serious obstacle to the advancement of ER, because long term planning is required by mobile robots navigating through unstructured environments. Planning requires the memory capabilities of RNNs, but it is uncertain if they are capable synthesising and retaining the long term memory required to plan. Alternatively, an approach similar to that briefly described in this thesis could be followed, where a separate process maintains a representation of the environment and neurocontrollers are evolved to operate on this information.

The framework presented in this thesis identifies a number of issues that are worthy of further research. Although solutions to some of these issues are not immediately obvious, strategies must be found to cope with these issues if NE is to find widespread application on practical RL problems.

8.3 Final remarks

The ER approach of evolving sensorimotor neurocontrollers is not suitable for application to small low-cost AUVs using current state-of-the-art methods. This is because it is impractical to evolve a single all-in-one controller to process the sensor data, control the vehicle dynamics as well as control the vehicle's high-level behaviour. None of the existing NE methods, including those that evolve NN architectures, have demonstrated the capability to solve such complex problems. However, it is doubtful that the answer to this issue lies solely with the development of more advanced NE methods. This is because more advanced methods do not decrease the difficulty of designing fitness functions to evolve such disparate capabilities nor ease the burden created by the requirement for very accurate simulation environments.

Rather than evolving sensorimotor neurocontrollers for small low-cost AUVs, it is far more practical to apply NE to RL problems within a hybrid control architecture. This way neurocontrollers can be evolved to control the vehicle's mid-level or high-level behaviour using preprocessed sensor data, while the vehicle's dynamics are controlled by a dedicated low-level vehicular controller. NE has shown promise for solving RL problems in robotics when applied in this manner. However, further research is required before NE can be recommended for routine application in the robotics industry.

In the years to come, the cost of building robotic platforms will continue to decrease. Yet, the difficulty of designing autonomous control systems for mobile robots will remain a barrier to their widespread application. Evolutionary methods do not enable the automatic design of robot control systems. Rather, they shift the effort of a human designer from the design of a solution to the design of a function that rewards a desired solution. Nevertheless, evolutionary methods can assist a human designer in solving control problems where the exact nature of the solution is unknown so long as they have realistic expectations and understand their limitations. Ultimately, the development of automatic processes for the synthesis of robot control systems remains an appealing, yet elusive, goal.

APPENDIX A

Inverted pendulum equations of motion

This appendix derives the equations of motion for the dual inverted pendulum system.

A.1 Nomenclature

x	Position of the cart (m)
θ_1	Angle of the long pole (rad)
θ_2	Angle of the short pole (rad)
m	Mass of the cart (kg)
m_1	Mass of the long pole (kg)
m_2	Mass of the short pole (kg)
I_1	Moment of inertia of the long pole $(\mathrm{kg} \cdot \mathrm{m}^2)$
I_2	Moment of inertia of the short pole $(\mathrm{kg} \cdot \mathrm{m}^2)$
$L_1 = 2l_1$	Length of the long pole (m)
$L_2 = 2l_2$	Length of the short pole (m)
F	Force applied to the cart (N)

A.2 Derivation

The equations of motion for the dual inverted pendulum system are derived in using Lagrange's equations. The general approach of Bogdanov (2004) for the jointed inverted pendulum system is followed. The dual inverted pendulum system is shown in Figure A.1.

The kinetic energy T of the dual inverted pendulum system consists of the kinetic energy

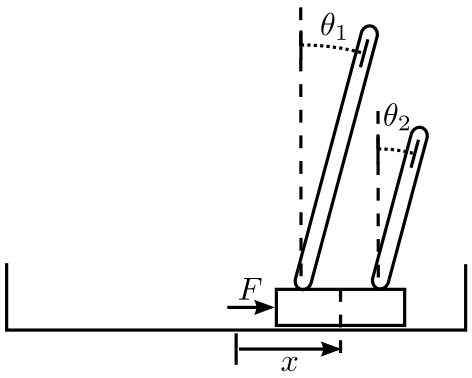


Figure A.1: Dual inverted pendulum system.

of the cart T_0 and the two poles T_1 and T_2. Thus:

$$T = T_0 + T_1 + T_2 \tag{A.1}$$

where:

$$T_0 = \frac{1}{2} m_0 \dot{x}^2 \tag{A.2}$$

$$T_1 = \frac{1}{2} I_1 \dot{\theta}_1^2 + \frac{1}{2} m_1 \left(\left(\dot{x} + l_1 \dot{\theta}_1 \cos\theta_1 \right)^2 + \left(l_1 \dot{\theta}_1 \sin\theta_1 \right)^2 \right) \tag{A.3}$$

$$T_2 = \frac{1}{2} I_2 \dot{\theta}_2^2 + \frac{1}{2} m_2 \left(\left(\dot{x} + l_2 \dot{\theta}_2 \cos\theta_2 \right)^2 + \left(l_2 \dot{\theta}_2 \sin\theta_2 \right)^2 \right) \tag{A.4}$$

The pendulum poles are assumed to uniform slender rods such that:

$$I_1 = \frac{m_1 L_1^2}{12} \tag{A.5}$$

$$I_2 = \frac{m_2 L_2^2}{12} \tag{A.6}$$

The potential energy V of the dual inverted pendulum system consists of the potential energy of the cart V_0 and the two poles V_1 and V_2. Thus:

$$V = V_0 + V_1 + V_2 \tag{A.7}$$

where:

$$V_0 = 0 \tag{A.8}$$

$$V_1 = m_1 g l_1 \cos\theta_1 \tag{A.9}$$

$$V_2 = m_2 g l_2 \cos\theta_2 \tag{A.10}$$

The Lagrange equations for the dual inverted pendulum system are:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{x}}\right) - \frac{\partial L}{\partial x} = F \tag{A.11}$$

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta_1}}\right) - \frac{\partial L}{\partial \theta_1} = 0 \tag{A.12}$$

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta_2}}\right) - \frac{\partial L}{\partial \theta_2} = 0 \tag{A.13}$$

where $L = T - V$ is the Lagrangian.

Following evaluation and collecting terms, the Lagrange equations can be expressed as:

$$\mathbf{D}[\ddot{x}, \ddot{\theta}_1, \ddot{\theta}_2]^{\mathrm{T}} + \mathbf{C}[\dot{x}, \dot{\theta}_1, \dot{\theta}_2]^{\mathrm{T}} + \mathbf{G} = \mathbf{H}u \tag{A.14}$$

where:

$$\mathbf{D} = \begin{bmatrix} m_0 + m_1 + m_2 & \frac{1}{2}L_1 m_1 \cos\theta_1 & \frac{1}{2}L_2 m_2 \cos\theta_2 \\ \frac{1}{2}L_1 m_1 \cos\theta_1 & \frac{1}{3}L_1^2 m_1 & 0 \\ \frac{1}{2}L_2 m_2 \cos\theta_2 & 0 & \frac{1}{3}L_2^2 m_2 \end{bmatrix} \tag{A.15}$$

$$\mathbf{C} = \begin{bmatrix} 0 & -\frac{1}{2}L_1 m_1 \dot{\theta}_1 \sin\theta_1 & -\frac{1}{2}L_2 m_2 \dot{\theta}_2 \sin\theta_2 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{A.16}$$

$$\mathbf{G} = \begin{bmatrix} 0 \\ -\frac{1}{2}gL_1 m_1 \sin\theta_1 \\ -\frac{1}{2}gL_2 m_2 \sin\theta_2 \end{bmatrix} \tag{A.17}$$

$$\mathbf{H} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \tag{A.18}$$

Or reformulated as an ordinary differential equation:

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \mathbf{0} & \mathbf{I} \\ \mathbf{0} & -\mathbf{D}^{-1}\mathbf{C} \end{bmatrix} \boldsymbol{x} + \begin{bmatrix} \mathbf{0} \\ -\mathbf{D}^{-1}\mathbf{G} \end{bmatrix} + \begin{bmatrix} \mathbf{0} \\ \mathbf{D}^{-1}\mathbf{H} \end{bmatrix} u \tag{A.19}$$

where $\boldsymbol{x} = [x, \theta_1, \theta_2, \dot{x}, \dot{\theta}_1, \dot{\theta}_2]^{\mathrm{T}}$ is the state vector.

References

Adams, B. (2006). *Evolutionary, developmental neural networks for robust robotic control*. PhD thesis, Department of Electrical Engineering and Computer Science, MIT, USA.

Alpaydin, E. (2004). *Introduction to Machine Learning*. MIT Press, USA.

Arkin, R. C. (1998). *Behavior-Based Robotics*. MIT Press, USA.

Bäck, T. (2000). *Evolutionary Computation 2: Advanced Algorithms and Operators*, chapter Self-adaptation, pages 188–210. CRC Press.

Baldassarre, G., Nolfi, S., and Parisi, D. (2003). Evolving mobile robots able to display collective behavior. *Artificial Life*, 9:255–267.

Barlow, G. J., Mattos, L. S., Grant, E., and Oh, C. K. (2005). Transference of Evolved Unmanned Aerial Vehicle Controllers to a Wheeled Mobile Robot. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2087–2092.

Beyer, H.-G. and Arnold, D. V. (2003). Qualms Regarding the Optimality of Cumulative Path Length Control in CSA/CMA-Evolution Strategies. *Evolutionary Computation*, 11(1):19–28.

Beyer, H.-G. and Deb, K. (2001). On self-adaptive features in real-parameter evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 5(3):250–270.

Beyer, H.-G. and Schwefel, H.-P. (2002). Evolution strategies - A comprehensive introduction. *Natural Computing*, 1(1):3–52.

Bogdanov, A. (2004). Optimal control of a double inverted pendulum on a cart. Technical Report CSE-04-006, Department of Computer Science and Electrical Engineering, OGI School of Science and Engineering, OHSU, USA.

Brooks, R. A. (1986). A Robust Layered Control System for a Mobile Robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23.

Brutzman, D. P. (1994). *A virtual world for an autonomous underwater vehicle*. PhD thesis, Naval Postgraduate School, USA.

Chellapilla, K. and Fogel, D. B. (2001). Evolving an expert checkers playing program without using human expertise. *IEEE Transactions on Evolutionary Computation*, 5(4):422–428.

Choset, H., Lynch, K. M., Hutchinson, S., Kantor, G., Burgard, W., Kavraki, L. E., and Thrun, S. (2005). *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, USA.

Cliff, D. and Miller, G. F. (1995). Tracking the Red Queen: Measurements of Adaptive Progress in Co-Evolutionary Simulations. In *Proceedings of the Third European Conference on Advances in Artificial Life*, pages 200–218.

Davidson, E. (2006). *The Regulatory Genome: Gene Regulatory Networks in Development and Evolution*. Academic Press.

De Nardi, R., Togelius, J., Holland, O. E., and Lucas, S. M. (2006). Evolution of Neural Networks for Helicopter Control: Why Modularity Matters. In *Proceedings of the IEEE Congress on Evolutionary Computation*.

Dubins, L. E. (1957). On Curves of Minimal Length with a Constraint on Average Curvature, and with Prescribed Initial and Terminal Positions and Tangents. *American Journal of Mathematics*, 79(3):497–516.

Dürr, P., Mattiussi, C., and Floreano, D. (2006). *Neuroevolution with Analog Genetic Encoding*, volume 4193 of *Lecture Notes in Computer Science*, chapter Parallel Problem Solving from Nature - PPSN IX, pages 671–680. Springer Berlin, Germany.

Eiben, A. and Smith, J. (2003). *Introduction to Evolutionary Computing*. Natural Computing Series. Springer, Germany.

Floreano, D., Dürr, P., and Mattiussi, C. (2008). Neuroevolution: from architectures to learning. *Evolutionary Intelligence*, 1(1):47–62.

Floreano, D. and Mondada, F. (1994). Automatic Creation of an Autonomous Agent: Genetic Evolution of a Neural Network Driven Robot. In Cliff, D., Husbands, P., Meyer, J.-A., and Wilson, S., editors, *3rd International Conference on Simulation of Adaptive Behavior (SAB'1994)*. MA: MIT Press. D. Cliff, P. Husbands, J.-A. Meyer, and S. Wilson (eds.).

Floreano, D. and Mondada, F. (1996). Evolution of homing navigation in a real mobile robot. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 26(3):396–407.

Floreano, D. and Urzelai, J. (2000). Evolutionary Robots with on-line self-organization and behavioral fitness. *Neural Networks*, 13(4-5):431–443.

Fogel, L. J. (1962). Autonomous Automata. *Industrial Research Magazine*, 4(2):14–19.

Fogel, L. J., Owens, A. J., and Walsh, M. J. (1966). *Artificial Intelligence Through Simulated Evolution*. John Wiley & Sons, USA.

Gallagher, M. and Ledwich, M. (2007). Evolving Pac-Man Players: Can We Learn from Raw Input? In *IEEE Symposium on Computational Intelligence and Games*, pages 282–287.

Geard, N. and Wiles, J. (2003). A gene regulatory network for cell differentiation in caenorhabditis elegans. In *The First Australian Conference on Artificial Life*.

Geva, S. and Sitte, J. (1992). A one neuron truck backer-upper. In *International Joint Conference on Neural Networks*, volume 2, pages 850–856.

Geva, S. and Sitte, J. (1993). A cartpole experiment benchmark for trainable controllers. *IEEE Control Systems Magazine*, 13(5):40–51.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Professional, USA.

Gomez, F., Schmidhuber, J., and Miikkulainen, R. (2006). Efficient Non-linear Control Through Neuroevolution. In *Machine Learning: ECML 2006*, Lecture Notes in Computer Science, pages 654–662, Germany. Springer Berlin.

Gomez, F., Schmidhuber, J., and Miikkulainen, R. (2008). Accelerated Neural Evolution through Cooperatively Coevolved Synapses. *Journal of Machine Learning Research*, 9:937–965.

Gomez, F. J. (2003). *Robust non-linear control through neuroevolution*. PhD thesis, Department of Computer Sciences, The University of Texas at Austin, USA.

Gomez, F. J. and Miikkulainen, R. (1999). Solving non-markovian control tasks with neuroevolution. In *Proceedings of the sixteenth international joint conference on artificial intelligence*, pages 1356–1361, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Gomez, F. J. and Miikkulainen, R. (2003). Active guidance for a finless rocket using neuroevolution. In *Proceedings of the the Genetic and Evolutionary Computation Conference (GECCO-2003)*.

Gomez, F. J. and Miikkulainen, R. (2004). Transfer of Neuroevolved Controllers in Unstable Domains. In *Genetic and Evolutionary Computation - GECCO 2004*, volume 3103 of *Lecture Notes in Computer Science*, pages 957–968, Germany. Springer Berlin.

Gruau, F., Whitley, D., and Pyeatt, L. (1996). A comparison between cellular encoding and direct encoding for genetic neural networks. In Koza, J. R., Goldberg, D. E., Fogel, D. B., and Riolo, R. L., editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 81–89, Stanford University, CA, USA. MIT Press.

Hansen, N. and Kern, S. (2004). Evaluating the CMA Evolution Strategy on Multimodal Test Functions. In *Eighth International Conference on Parallel Problem Solving from Nature PPSN VIII*, pages 282–291, Germany. Springer Berlin.

Hansen, N. and Ostermeier, A. (1996). Adapting Arbitrary Normal Muation Distributions in Evolution Strategies: The Covariance Matrix Adaptation. In *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation*, pages 312–317.

Hansen, N. and Ostermeier, A. (2001). Completely Derandomized Self-Adaptation in Evolution Strategies. *Evolutionary Computation*, 9(2):159–195.

Harvey, I., Husbands, P., and Cliff, D. (1993). Issues in evolutionary robotics. In *Proceedings of the second international conference on From animals to animats 2: simulation of adaptive behavior: simulation of adaptive behavior*, pages 364–373.

Haykin, S. (1999). *Neural networks: A Comprehensive Foundation*. Prentice Hall, Germany, 2nd edition.

Holland, J. H. (1992). *Adaptation in natural and artificial systems*. MIT Press, USA, 2nd edition.

Hornby, G. S., Takamura, S., Yokono, J., Hanagata, O., Fujita, M., and Pollack, J. (2000a). *Evolvable Systems: From Biology to Hardware*, volume 1801 of *Lecture Notes in Computer Science*, chapter Evolution of Controllers from a High-Level Simulator to a High DOF Robot, pages 80–89. Springer Berlin, Germany.

Hornby, G. S., Takamura, S., Yokono, J., Hanagata, O., Yamamoto, T., and Fujita, M. (2000b). Evolving robust gaits with AIBO. In *IEEE International Conference on Robotics and Automation*, volume 3, pages 3040–3045.

Igel, C. (2003). Neuroevolution for reinforcement learning using evolution stategies. In *The 2003 Congress on Evolutionary Computation*, volume 4, pages 2588–2595.

Jacobi, N., Husbands, P., and Harvey, I. (1995). Noise and the Reality Gap: The Use of Simulation in Evolutionary Robotics. In *Proceedings of the Third European Conference on Advances in Artificial Life*, pages 704–720.

Kassahun, Y. (2006). *Towards a unified approach to learning and adaptation*. PhD thesis, Inst. f. Informatik u. Prakt. Math. der Christian-Albrechts-Universität zu Kiel, Germany.

Kern, S., Müller, S. D., Hansen, N., Büche, D., Ocenasek, J., and Koumoutsakos, P. (2004). Learning Probability Distributions in Continuous Evolutionary Algorithms - A Comparative Review. *Natural Computing*, 3(1):77–112.

Klockgether, J. and Schwefel, H. (1970). Two-phase nozzle and hollow core jet experiments. In Elliott, D., editor, *Engineering Aspects of Magnetohydrodynamics*, pages 141–148.

Kondo, T., Ishiguro, A., Tokura, S., Uchikawa, Y., and Eggenberger, P. (1999). Realization of robust controllers in evolutionary robotics: a dynamically-rearranging neural network approach. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 1.

Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, USA.

LaValle, S. M. (2006). *Planning algorithms*. Cambridge University Press, USA.

Lucas, S. M. (2005). Evolving a Neural Network Location Evaluator to Play Ms. Pac-Man. In *IEEE Symposium on Computational Intelligence and Games*, pages 203–210.

Lucas, S. M. and Kendall, G. (2006). Evolutionary computation and games. *IEEE Computational Intelligence Magazine*, 1(1):10–18.

Matarić, M. and Cliff, D. (1996). Challenges in evolving controllers for physical robots. *Robotics and Autonomous Systems*, 19(1):67–83.

Mattiussi, C. (2005). *Evolutionary synthesis of analog networks*. PhD thesis, Laboratory of Intelligent Systems, Institute of System Engineering, Ecole Polytechnique Fédérale de Lausanne (EPFL), Switzerland.

Meyer-Nieberg, S. and Beyer, H.-G. (2007). Self-Adaptation in Evolutionary Algorithms. In *Parameter Setting in Evolutionary Algorithms*, pages 47–75.

Mondada, F., Franzi, E., and Ienne, P. (1994). Mobile robot miniaturisation: A tool for investigation in control algorithms. In *The 3rd international symposium on experimental robotics*, pages 501–513, London, UK. Springer-Verlag.

Moriarty, D. E. (1997). *Symbiotic evolution of neural networks in sequential decision tasks*. PhD thesis, Department of Computer Sciences, The University of Texas at Austin, USA.

Moriarty, D. E. and Mikkulainen, R. (1996). Efficient reinforcement learning through symbiotic evolution. *Machine Learning*, 22:11–32.

Nelson, A. L. (2003). *Competitive relative performance and fitness selection for evolutionary robotics*. PhD thesis, North Carolina State University, USA.

Nelson, A. L. and Grant, E. (2006). Using direct competition to select for competent controllers in evolutionary robotics. *Robotics and Autonomous Systems*, 54(10):840–857.

Nguyen, D. and Widrow, B. (1989). The truck backer-upper: an example of self-learning in neural networks. In *International Joint Conference on Neural Networks*, volume 2, pages 357–363, USA.

Nolfi, S. (1997). Evolving non-trivial behaviors on real robots: A garbage collecting robot. *Robotics and Autonomous Systems*, 22(3):187–198.

Nolfi, S. (2002). Power and the limits of reactive agents. *Neurocomputing*, 49:119–145.

Nolfi, S. and Floreano, D. (1998). Coevolving Predator and Prey Robots: Do "Arms Races" Arise in Artificial Evolution? *Artificial Life*, 4(4):311–335.

Nolfi, S. and Floreano, D. (1999). Learning and evolution. *Autonomous Robots*, 7(1):89–113.

Nolfi, S. and Floreano, D. (2000). *Evolutionary Robotics: The Biology, Intelligence and Technology of Self-Organizing Machines*. The MIT Press, USA.

Nolfi, S., Floreano, D., Miglino, O., and Mondada, F. (1994). How to Evolve Autonomous Robots: Different Approaches in Evolutionary Robotics. In Brooks, R. A. and Maes, P., editors, *4th International Workshop on Artificial Life*, USA. MIT Press. R. A. Brooks and P. Maes (eds.).

Oh, C. K. and Barlow, G. J. (2004). Autonomous Controller Design for Unmanned Aerial Vehicles using Multi-objective Genetic Programming. In *Proceedings of the 2004 Congress on Evolutionary Computation*, pages 1538–1545.

Ostermeier, A. and Hansen, N. (1999). An evolution strategy with coordinate system invariant adaptation of arbitrary normal mutation distributions within the concept of mutative strategy

parameter control. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 902–909.

Panait, L. and Luke, S. (2005). Cooperative Multi-Agent Learning: The State of the Art. *Autonomous Agents and Multi-Agent Systems*, 11(3):387–434.

Prestero, T. (2001). Verification of a six-degree of freedom simulation model for the REMUS autonomous underwater vehicle. Master's thesis, University of California at Davis, Joint Program in Applied Ocean Science and Engineering, USA.

Quinn, M. (2000). Evolving co-operative homogeneous multi-robot teams. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 3, pages 1798–1803.

Rechenberg, I. (1973). *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog.

Reil, T. (1999). Dynamics of gene expression in an artificial genome - implications for biological and artificial ontogeny. In *Proceedings of the 5th European Conference on Artificial Life*, pages 457–466.

Rudolph, G. (1992). On correlated mutations in evolution strategies. In Männer, R. and Manderick, B., editors, *Parallel Problem Solving from Nature 2*, pages 105–114.

Saravanan, N. and Fogel, D. (1995). Evolving Neural Control Systems. *IEEE Expert*, 10(3):23–27.

Schwefel, H.-P. (1975). *Evolutionsstrategie und numerische Optimierung*. PhD thesis, Technische Universität, Berlin, Germany.

Shkel, A. M. and Lumelsky, V. (2001). Classification of the Dubins set. *Robotics and Autonomous Systems*, 34(4):179–202.

Siebel, N. T., Krause, J., and Sommer, G. (2007). Efficient learning of neural networks with evolutionary algorithms. In *Proceedings of the 29th Annual Symposium of the German Association for Pattern Recognition (DAGM2007)*, pages 466–475, Berlin, Germany. Springer Verlag.

Sit, Y. F. and Miikkulainen, R. (2005). Learning basic navigation for personal satellite assistant using neuroevolution. In *Proceedings of the Genetic and Evolutionary Computation Conference*.

Souéres, P. and Boissonnat, J.-D. (1998). *Robot Motion Planning and Control*, volume 229 of *Lecture notes in control and information sciences*, chapter Optimal trajectories for nonholonomic mobile robots. Springer-Verlag, Germany.

Stanley, K. O. (2004). *Efficient Evolution of Neural Networks through Complexification*. PhD thesis, Department of Computer Sciences, The University of Texas at Austin, USA.

Stanley, K. O., Bryant, B. D., and Miikkulainen, R. (2005). Real-time neuroevolution in the NERO video game. *IEEE Transactions on Evolutionary Computation*, 9(6):653–669.

Stanley, K. O. and Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127.

Stanley, K. O. and Miikkulainen, R. (2003). A Taxonomy for Artificial Embryogeny. *Artificial Life*, 9(2):93–130.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press, USA.

Taylor, M. E., Whiteson, S., and Stone, P. (2006). Comparing evolutionary and temporal difference methods in a reinforcement learning domain. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 1321–1328.

Tempo, R., Calafiore, G., and Dabbene, F. (2004). *Randomized Algorithms for Analysis and Control of Uncertain Systems*. Communications and Control Engineering. Springer, Germany.

Thrun, S., Burgard, W., and Fox, D. (2005). *Probabilistic Robotics*. MIT Press, USA.

Togelius, J., Lucas, S. M., and De Nardi, R. (2007). *Advanced Intelligent Paradigms in Computer Games*, volume 71, chapter Computational Intelligence in Racing Games, pages 36–69. Springer Berlin, Germany.

Tuci, E., Quinn, M., and Harvey, I. (2002). Evolving fixed-weight networks for learning robots. In *Proceedings of the 2002 Congress on Evolutionary Computation*, volume 2.

Urzelai, J. and Floreano, D. (2001). Evolution of Adaptive Synapses: Robots with Fast Adaptive Behavior in New Environments. *Evolutionary Computation*, 9(4):495–524.

Watson, R., Ficici, S., and Pollack, J. (1999). Embodied evolution: distributing an evolutionary algorithm in a population of robots. In *Proceedings of the 1999 Congress on Evolutionary Computation*, volume 1, pages 335–342.

Wharington, J. M. (2003). Odessa: Software for rigid body dynamics simulation. Technical Report 2003/42811/1, Defence Science and Technology Organisation (DSTO), Department of Defence, Maribyrnong, Australia.

Whitley, D., Dominic, S., Das, R., and Anderson, C. W. (1993). Genetic reinforcement learning for neurocontrol problems. *Machine Learning*, 13(2-3):259–284.

Widrow, B. (1987). The original adaptive neural net broom-balancer. In *Proceedings of the IEEE International Symposium Circuits & Systems*, pages 351–357.

Widrow, B. and Smith, F. W. (1964). Pattem recognizing control systems. In *Computer and Information Sciences (COINS) Proceedings*, pages 288–317.

Wieland, A. P. (1991). Evolving neural network controllers for unstable systems. In *IJCNN-91-Seattle International Joint Conference on Neural Networks*, volume 2, pages 667–673.

Wolpert, D. H. and Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82.

Wolpert, D. H. and Macready, W. G. (2005). Coevolutionary free lunches. *IEEE Transactions on Evolutionary Computation*, 9(6):721–735.

Yao, X. (1999). Evolving Artificial Neural Networks. *Proceedings of the IEEE*, 87(9):1423–1447.

Yuh, J. (2000). Design and Control of Autonomous Underwater Robots: A Survey. *Autonomous Robots*, 8(1):7–24.

Zufferey, J.-C. (2005). *Bio-inspired vision-based flying robots*. PhD thesis, Ecole Polytechnique Fédérale de Lausanne (EPFL), Switzerland.